STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A
COMPUTER NETWORK

THESIS

Nathan C. Stuckey, CAPTAIN, USAF

AFIT/GE/ENG/07-24

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GE/ENG/07-24

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

THESIS

Nathan C. Stuckey, B.S.E.E

CAPTAIN, USAF

March 2007
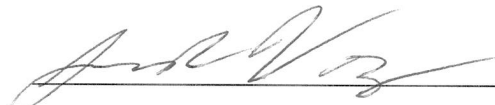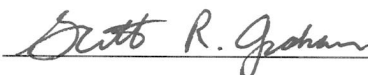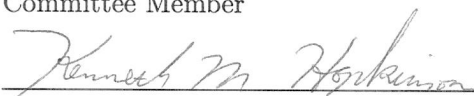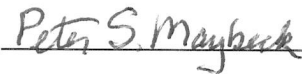
AFIT/GE/ENG/07-24

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A
COMPUTER NETWORK

Nathan C. Stuckey, B.S.E.E

CAPTAIN, USAF

Approved:

_____   _____
Lt Col Juan R. Vasquez   27 Feb 07
Thesis Advisor   Date

_____   _____
Maj Scott R. Graham   27 Feb 07
Committee Member   Date

_____   _____
Dr Kenneth M. Hopkinson   27 Feb 07
Committee Member   Date

_____   _____
Dr Peter S. Maybeck   27 Feb 07
Committee Member   Date

AFIT/GE/ENG/07-24

Abstract

An extended Kalman filter is used to estimate size and packet arrival rate of network queues. These estimates are used by an LQG *steady state* linear perturbation PI controller to regulate queue size within a computer network. This research presents the derivation of the transient queue behavior for a system with Poisson traffic and exponential service times. This result is then validated for ideal traffic using a network simulated in OPNET. A more complex OPNET model is then used to test the adequacy of the transient queue size model when non-Poisson traffic is combined. The extended Kalman filter theory is presented and a network state estimator is designed using the transient queue behavior model. The equations needed for the LQG synthesis of a steady state linear perturbation PI controller are presented. These equations are used to develop a network queue controller based on the transient queue model. The performance of the network state estimator and network queue controller was investigated and shown to provide improved control when compared to other simplistic control algorithms.

Table of Contents

List of Figures

viii

List of Tables

STOCHASTIC ESTIMATION AND CONTROL OF QUEUES WITHIN A COMPUTER NETWORK

I. Introduction

While the current routing and congestion control algorithms in use today may be sufficient for networks with a relatively static topology, these algorithms may not be sufficient for the dynamically changing networks that will be found on the battlefield in the near future. It is easy to envision many instances of rapidly changing networks in use on the battlefield as depicted in Figure 1.1. For example, a group of unmanned aerial vehicles utilizing laser communications would exhibit quickly changing network characteristics. Cloud interference would cause communication links to be broken constantly, and the rapid movement of aircraft would create a network topology continuously in flux. In these cases, it is expected that dynamic routing and congestion control algorithms based on stochastic estimation and control would provide superior results to more traditional algorithms.

The goal of this research is to improve the process of information delivery on computer networks. Current mainstream routing and congestion control algorithms do not take the probabilistic nature of computer networks into account. The past performance of a computer network can be determined by observing measurements taken at various nodes of the network. However, since these measurements cannot instantly be available at a central node and these measurements will contain a certain amount of error, it is impossible to know the current state of the network deterministically. If the stochastic nature of a computer network can be modeled accurately, the current state of the network can be estimated and the future state can be predicted. Being able to estimate the current and future state of the network would allow for optimal routing and congestion control algorithms to be developed. The goal of this research is to model a computer network stochastically and to estimate and predict the metrics of interest. This information can then be used to develop optimal network control algorithms.
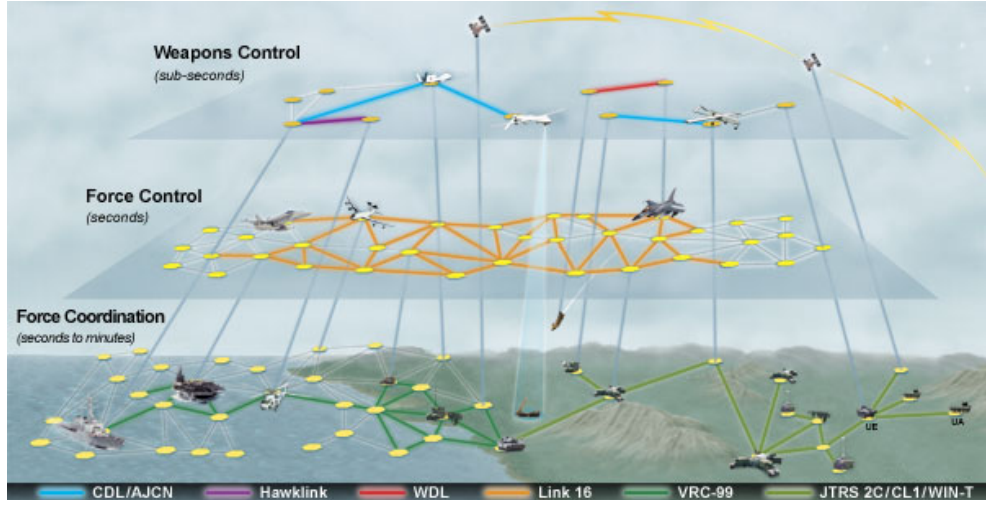
Figure 1.1   Representative battlefield network [2]

A network control algorithm can be developed based on a feedback controller. Figure 1.2 shows the basic structure of a feedback controller [18]. The goal of a feedback controller is to provide the control $\mathbf{u}$ to the dynamic system such that the controlled variables $\mathbf{y_c}$ match the reference signal $\mathbf{y_d}$ as closely as possible. Dynamics disturbances $\mathbf{n}$ also effect the dynamic system, usually in an undesirable way. In order to observe these disturbances, measurements $\mathbf{z}$ of the dynamic system are taken and are fed back to the controller. These measurements may correspond to the controlled variables, but in many cases not all of the controlled variables will be measured. The control $\mathbf{u}$ is computed based on the feedback the measurements provide about the state of the dynamic system. The measurements in general are not perfect due to measurement corruptions $\mathbf{n_m}$.

The temperature control system of a building is an example of a feedback control system. The dynamic system is the environment in the building and the controlled variable is the temperature of the building. It is desired to maintain the temperature of the building at the reference point set by the inhabitants of the building. The controller computes the control inputs to be sent to the furnace, air conditioner, or air duct valves based on the measurement of the temperature provided by a thermostat. Dynamics disturbances come from a number of sources like the effect of the outside temperature and heat added by lighting.
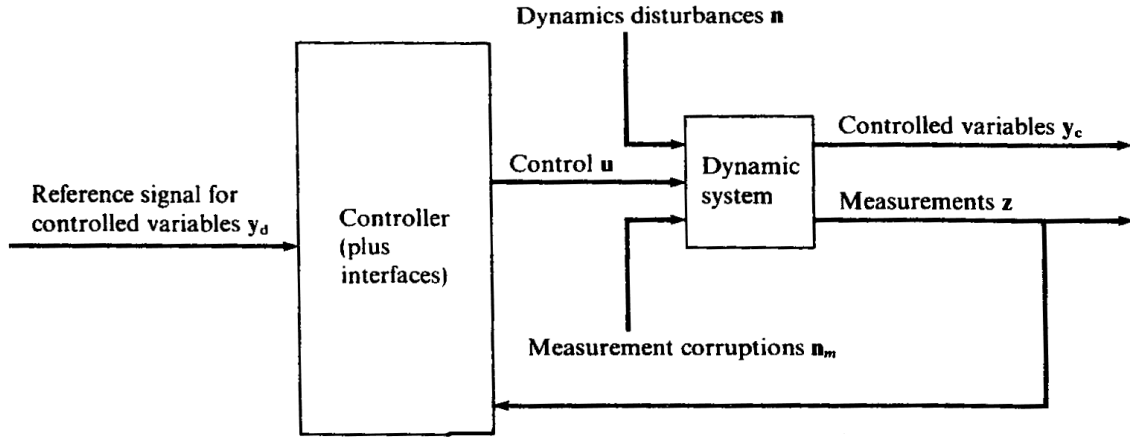
Figure 1.2   Controlled system configuration, from Maybeck [18]

A stochastic controller can be developed which takes the unknown quantities, dynamics disturbances and measurement corruptions into account to provide better control. Figure 1.3 shows the basic structure of a stochastic LQG (Linear-Quadratic-Gaussian) controller, to be discussed fully in Chapter III. The stochastic controller uses a Kalman filter to estimate the state of the system $\hat{\mathbf{x}}$ based on the measurements and previous control inputs. Optimal control inputs $\mathbf{u}^*$ are computed based on these state estimates and the optimal controller gains.

A control system can be designed in which the dynamic system is a computer network. The dynamic system includes components of the computer network, including computers, routers, data links, servers, switches, and hubs. The controlled variables consist of the performance metrics of interest such as network delay, data throughput, and congestion levels. To provide the desired service, the network controller will compute various control inputs, to include sending rates, routing table entries, optimal data packet sizes, etc.

Queues are one of the major components found in computer networks. In general, computer networks can be thought of as queuing networks. To demonstrate the feasibility of applying stochastic control theory to computer networks, a controller will be developed in this document to regulate queue sizes by controlling the packet arrival rate to the network queues. As part of this controller, a Kalman filter will be developed to estimate the size of a network queue and the total packet arrival rate to a network queue, given sample data

1-3
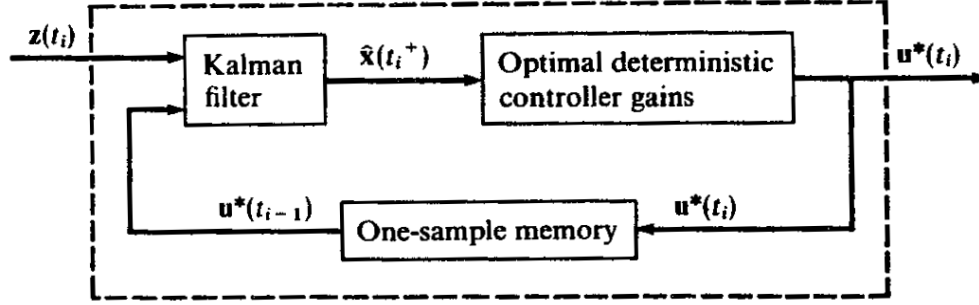
Figure 1.3   Overall structure of LQG controller, from Maybeck [18]

measurements of the queue size. One of the major research efforts needed to accomplish the above vision is the development of the state space model for the computer network of interest to be used by the Kalman filter. The following chapter will further motivate the need for stochastic control of computer networks and explore the efforts that have been made in this largely unexplored field. In Chapter III, the derivation of the transient network queue model is presented, stochastic estimation and control theory is presented, and a network estimator and queue controller are designed. In Chapter IV, the transient queue model is verified and the performance of the network estimator and queue controller are demonstrated. The thesis is concluded in Chapter V.

II. Literature Review

2.1 Introduction

For a computer network to function optimally, active feedback control mechanisms are necessary. Various routing and flow control algorithms control both the path information travels and the rate at which it is sent over the network. Knowledge of the state of the network is an essential element of the network control algorithms. Complex computer networks require sophisticated algorithms to provide the advanced control needed to meet the requirements of modern day applications optimally.

2.2 Network Tomography

2.2.1 Network Tomography Background. Network tomography is an emerging field in which the state of the network is inferred based on measurements. In general, a model is developed that sufficiently describes primary characteristics of the network. Measurements are then taken that either directly or indirectly correspond to network parameters. Finally, statistical inference techniques are then employed to estimate the actual state of the network.

Traditionally, it has been viewed as infeasible to measure the state of the network core directly [8]. Due to the complex and decentralized nature of computer networks, a coordinated measurement framework is very challenging to implement. The cost of adding new hardware coupled with the inter-operablity concerns of new hardware additions has prevented a coherent measurement framework from existing in computer networks. In the past, the added computational burden combined with the added bandwidth needed to collect data from within the network has been considered impractical.

Because of the reasons presented, most network measurement schemes have existed on the edges of the network instead of adding complexity to the network core. In these cases, the state of the network core must be inferred from the information obtained at the edges without coordination from the network core. These measurement either take the form of passive traffic measurements or active probe measurements [8]. Passive measurements

observe the existing local traffic, while active probe measurements insert small amounts of data into the network in order to gain knowledge of the network state.

As the bandwidth increases in modern computer networks, and the components of the computer network have an increasing amount of computation resources, active measurements of the network core are becoming possible. A notable example is explicit congestion notification (ECN), in which interior routers notify the senders on the edges of the network of congestion [9]. Also, both the multiprotocol label switching research presented by Anjali, Scoglio and Cavalcante de Oliverira [1] and the traffic matrix tracking presented by Soule, Salamatian, Nucci, and Taft [22], involve measurements taken within the network core.

Once core measurements are obtained, statistical inference techniques are used to analyze the measurement data in order to learn as much as possible about the state of the network. Numerous statistical techniques have been used including complexity-reducing hierarchical statistical models, moment- and likelihood-based estimation, expectation-maximization and Markov chain Monte Carlo algorithms [8].

One statistical technique that has had limited use in the network tomography field is the Kalman filter. As seen in Figure 2.1, the Kalman filter is fed measurements from the system of interest and produces an estimate of the system state. Within the Kalman filter, the system is modeled as a set of differential equations in the continuous-time case or as a set of difference equations in the case of a discrete-time system. The system model is used to propagate the estimate of the state of the system forward in time until a measurement is received. At this point, the system state attained from the measurement is compared to the estimate of the system state and combined in a optimal manner. Neither the estimate of the system nor the measurement can be trusted to be exact, because no model is perfect and no measurement is without error. The Kalman filter must know the amount of uncertainty associated with both the system model and measurements. The Kalman filter uses this knowledge of uncertainty to calculate how much weight to give the system state estimate versus the new information given by a measurement. Since the Kalman filter has knowledge of the system through the system model, a more exact estimate of the system state can be deduced than from the measurements alone.
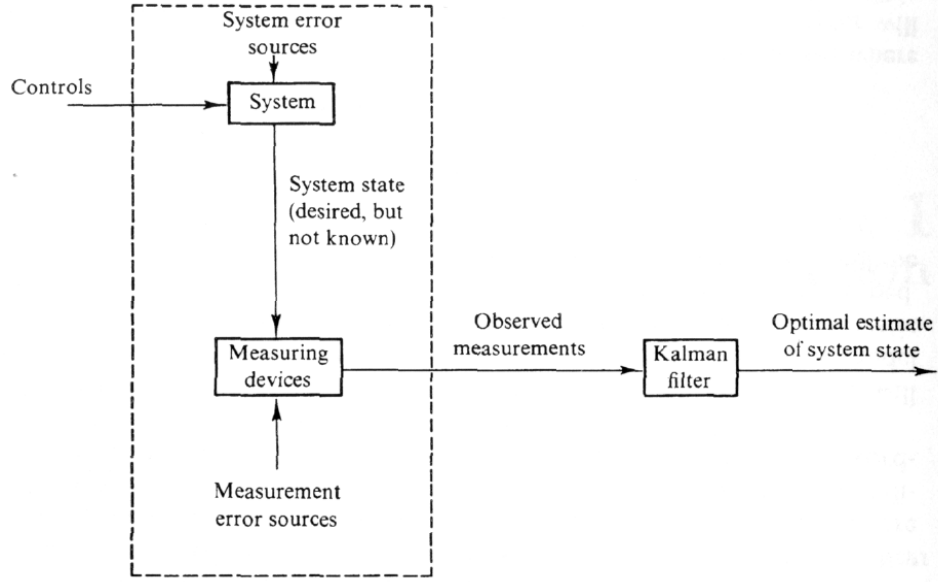
Figure 2.1   Typical Kalman filter application, from Maybeck [16]

2.2.2   Kalman Filter Based Network Tomography Examples.   Since network data is transmitted in arbitrarily sized packets, the Kalman filter has been used to estimate the optimal packet size in a wireless network in order to maximize channel efficiency [21]. In a wireless network, there is an overhead cost associated with sending each packet. Therefore if the packet size is too small, efficiency is decreased due to excessive overhead cost. However, since a transmission error will cause the entire contents of a packet to be lost, a large packet size will result in much data loss when an error in the wireless channel occurs, and efficiency is reduced. These two factors compete against each other, and the optimal packet size is a function of the bit-error-rate. If a large number of errors occur, small packets are desirable, but if there are fewer errors, larger packet sizes are more efficient. It was shown that a Kalman filter was able to estimate the optimal packet size with two orders of magnitude less estimation error than the more traditional moving average method [21].

Bletsas also demonstrated that the Kalman filter could be used for time synchronization between a client and a server which has an accurate measurement of "true" time, for example, by means of a Global Positioning System source [4]. Messages were transmitted

between the server and the client using the network time protocol (NTP) format. The Kalman filter was executed at the client side to provide the client with an accurate estimate of "true" time using the NTP messages as input. When the queuing delay difference between two consecutive NTP messages was a Gaussian random variable, the Kalman filtering method was found to be optimal. When the cross traffic was modeled as self-similar (long-range dependant), the Kalman filter outperformed all other methods except for linear programming. However, the Kalman filter still performed very well and provided a good trade-off in terms of performance across varying traffic scenarios.

Another use of the Kalman filter within the network tomography field is as part of a multiprotocol label switching (MPLS) network management technique in order to provide guaranteed quality of service for multimedia traffic [1]. In a multiprotocol label switching network, flows are set up between the source and destination by way of intermediate routers. Flows are set up and torn down based on the traffic demand. When a new flow is set up, a capacity allocation algorithm must reserve the necessary bandwidth from the intermediate routers. The algorithm must observe the network and choose a network path that has enough available bandwidth and also meets some kind of optimality criteria. In the research presented by [1], a Kalman filter is used to estimate the number of flows utilizing a path in order to discover the amount of extra capacity available. All of the flows in the network are assumed to have a constant data rate. The aggregate traffic is measured at the routers; and a Kalman filter is used due to the presence of measurement noise. The authors of the paper argue that the actual traffic measurements are noisy because the data rate must be deduced from sample data measurements and that even when the router can be accessed to provide detailed information, noise is still present due to delays and errors in the transfer of information. The Kalman filter is used to reduce this error and provide an accurate estimate of the number of flows upon which the capacity allocation algorithm can make decisions on how to route the flows in the network.

Finally, Kalman filters have been used to monitor origin-destination flows in a large network [22]. By measuring the amount of aggregate traffic on each router in the network, the authors were able to estimate the amount of traffic flowing between any origin-destination pair by way of a Kalman filter, given that the topology and routing tables

of the network are known. Since the authors were using a Kalman filter, it was possible to predict future network behavior. Additionally, the Kalman filter provided confidence intervals for the estimates and predictions. The authors were able to validate their research on Sprint's European IP backbone. The authors theorize that, due to the estimate and predictive capability of the Kalman filter along with its ability to provide confidence intervals, their setup would be useful for anomaly detection, including intruder threats and hardware failure.

2.3   Background TCP

Flow control in computer networks is one area that can benefit from advances in network tomography. Flow control ensures that the sender does not overwhelm the receiver or congest the network. To calculate the proper rate in which to send data on a computer network, the flow control algorithm must have some knowledge of the network state. Flow control within computer networks is mainly handled by the transmission control protocol (TCP). To better understand how TCP can benefit from stochastic estimation and control theory, a simple explanation of TCP based on Kurose and Ross's "Computer Networking" text [13] is presented below.

TCP provides a reliable, connection-oriented service to applications. The goal of TCP is to provide a mechanism for applications to communicate with each other over an unreliable computer network. TCP provides three main services to the application: transport-layer multiplexing and demultiplexing, reliable data transfer, and congestion control. Multiplexing is the process of getting data from the various applications at a sender, placing the data into packets (or segments as TCP packets are commonly called), and giving the segments to the computer network to deliver to the receiver. Once the data segments arrive at the receiver, the information is demultiplexed and given to the correct application. For more details into how this works, refer to [13, 24]. When an application desires to communicate with another application at a remote location, TCP sets up a connection between the two applications. This connection is bi-directional and can be closed by either application when communications end. Again, for more information on how TCP sets up and tears down connections, refer to [13, 24].

To send segments over the network, TCP hands the segments to the Internet Protocol (IP). IP is responsible for delivering the segments over the internet to the correct receiver. IP makes its "best effort" to deliver the segments over an unreliable network; however, no guarantees are made. Since IP cannot guarantee the delivery of segments, TCP must implement a reliable delivery mechanism. This is achieved through the use of acknowledgments (ACKs). As seen in Figure 2.2a, once the segment arrives at the receiver, an ACK is sent in order to notify the sender that the segment has been received. In the simplest case, the next segment would not be sent until the ACK is received (this method of stop-and-wait is how the very early versions of TCP worked). In order to increase efficiency, pipelining is used in which multiple unacknowledged segments are allowed to be in transit. This can be seen in Figure 2.2b in which three unacknowledged segments are allowed to exist.

In order for multiple unacknowledged segments to exist in transit, it is necessary to identify each individual segment in some manner to acknowledge which segments have been received. This is accomplished through the use of sequence numbers, which uniquely identify each segment in a TCP connection. TCP views the data stream between applications as a continuous ordered stream of bytes. Instead of assigning a sequential number to each packet, the byte-stream number of the first byte in the segment is used as the sequence number. This process is seen in Figure 2.3.

In the figure, a 500,000 byte file is divided into 1000 byte segments, which results in a total of 500 segments. The first segment is assigned sequence number 0, the second segment is assigned sequence number 1000, the third segment is assigned sequence number 2000, and so on. A TCP connection is bi-directional i.e., data is sent both directions. When a segment is acknowledged, the ACK will attempt to piggyback on a data packet traveling in the opposite direction. The ACK contains the sequence number that the sender is now ready to receive. This process is shown for a telnet session in Figure 2.4. The user types a single letter "C", which is sent to Host B as a single byte ASCII letter. Notice that Host A sends the single byte of data in a packet with a sequence number of 42, and that an ACK is piggybacked on the data segment in order to let Host B know it is ready for sequence number 79. In a telnet session, the received data is echoed back; so, the letter "C" is sent

a. Stop-and-wait operation

b. Pipelined operation
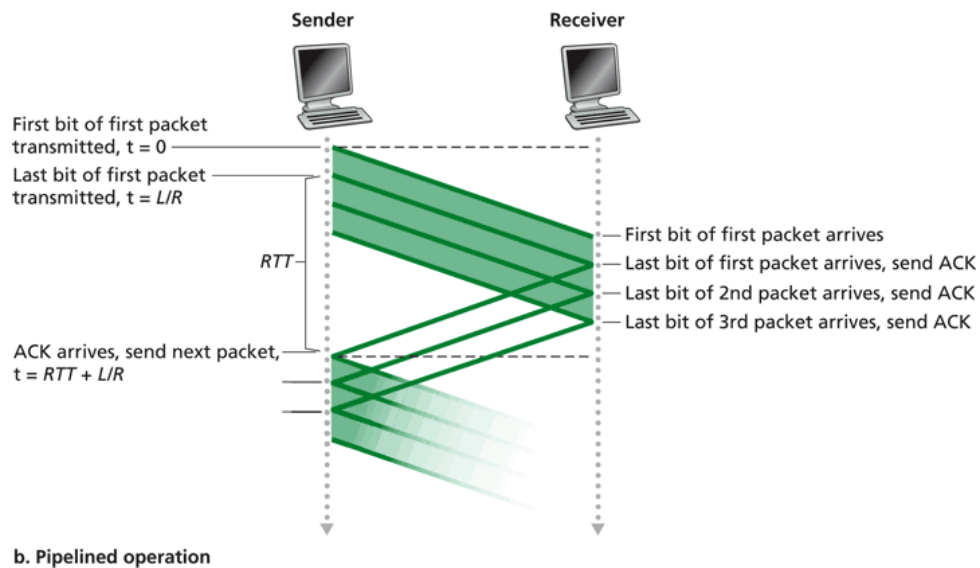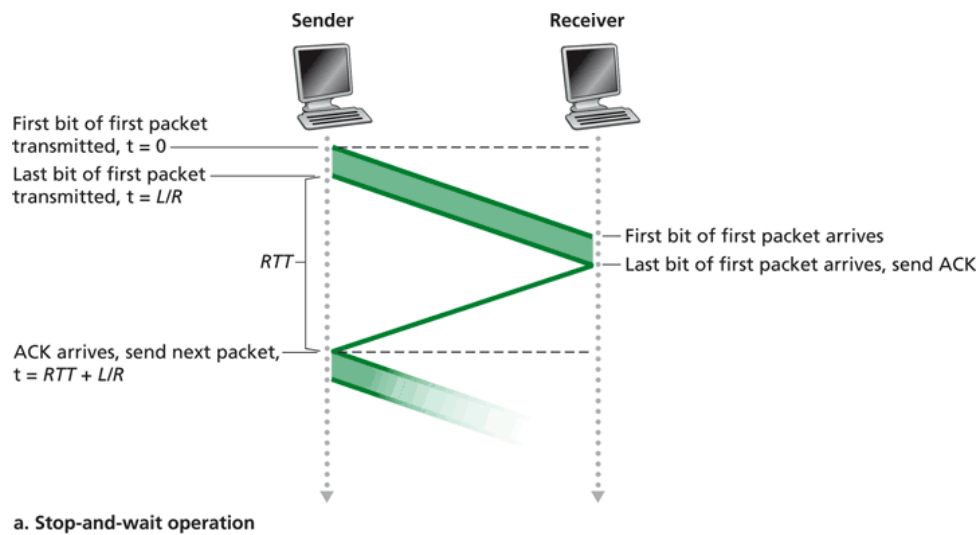
Figure 2.2   Stop-and-wait and pipelined sending, from Kurose and Ross [13]
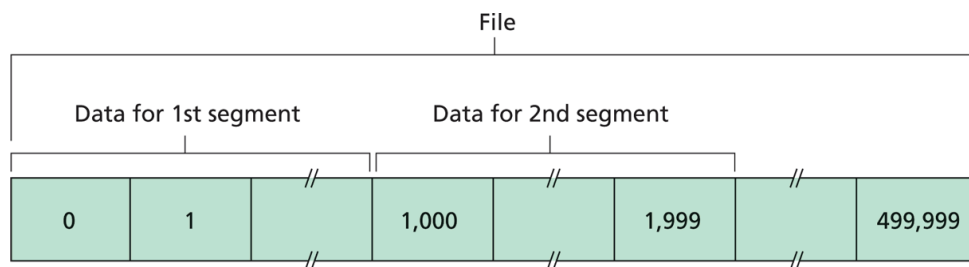
Figure 2.3   Dividing file data into TCP segments, from Kurose and Ross [13]
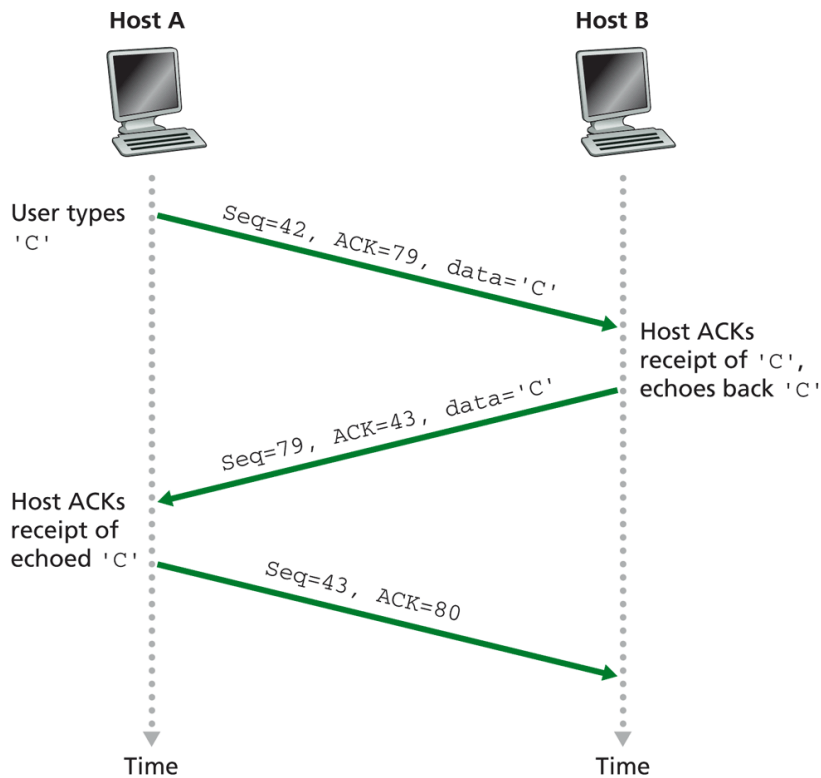
Figure 2.4   Sequence and acknowledgement numbers for a simple Telnet application over TCP, , from Kurose and Ross [13]

back to Host A with a sequence number of 79. Since the previous packet contained one byte of data, the ACK number is 43. Finally, Host A acknowledges that it received the data from Host B. Notice that in this case the ACK does not piggyback a data segment.

When a data segment is sent, a timer is started. If the sender does not receive an ACK for the segment after a certain amount of time, the timer expires and the segment is resent. In order to pick the length of the timer, the round trip time (RTT) is estimated. The RTT corresponds to the time from when the segment is sent to when the ACK is received. The timer is set to the current estimate of RTT plus a buffer to insure that the timer does not expire too early and unnecessarily resend a packet. Once the ACK is received, the timer is deleted. For more information on how the RTT is estimated or how the retransmission timer is calculated, refer to [13, 24]. In Figure 2.5, the retransmission time is demonstrated; notice that a lost ACK can also result in a retransmission.
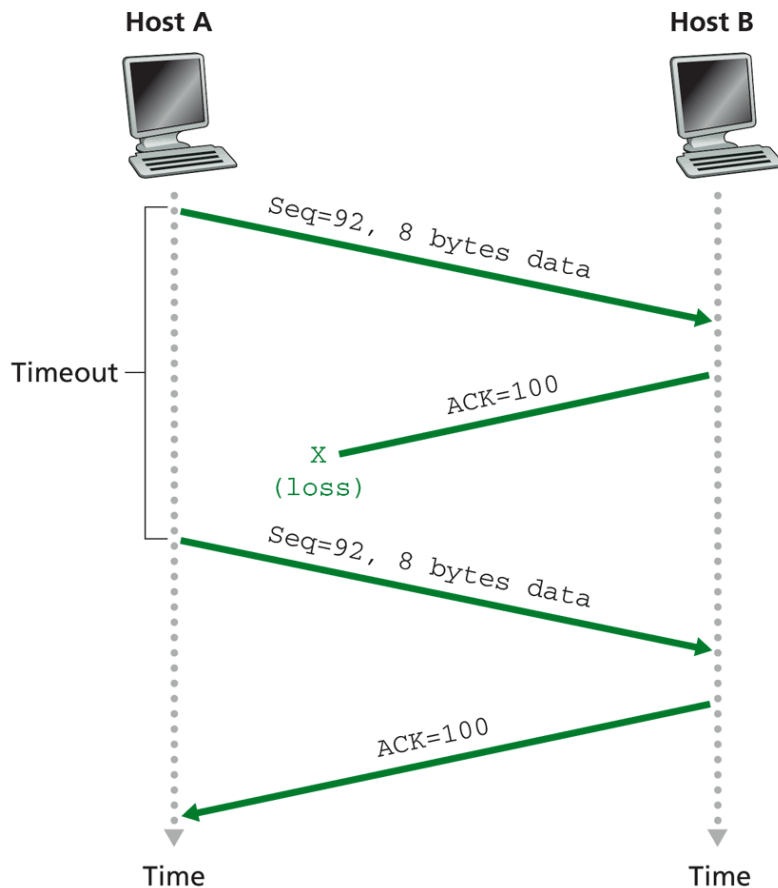
Figure 2.5   Retransmission due to a lost acknowledgment, from Kurose and Ross [13]

TCP uses cumulative acknowledgments, which means that TCP acknowledges up to the first missing byte in the stream. For example, in Figure 2.3, if segments 0 and 1000 have already been received and ACKed and then segment 3000 arrives, the ACK number sent will be 2000. The sender will then resend segment 2000 and since segment 3000 has already been received, the ACK number sent will be 4000 to avoid the unnecessary retransmission of segment 3000. Cumulative acknowledgments can also help avoid the retransmission of segments when the ACK has been lost in some cases, as seen if Figure 2.6. Since the cumulative acknowledgement states that Host B is ready for byte 120, segment 92 is not resent even though an ACK for that segment was not received. In addition to timeouts, TCP can also detect the presence of a lost segment through duplicate acknowledgments. Because of the pipelining feature discussed above, when a segment is lost, there are usually
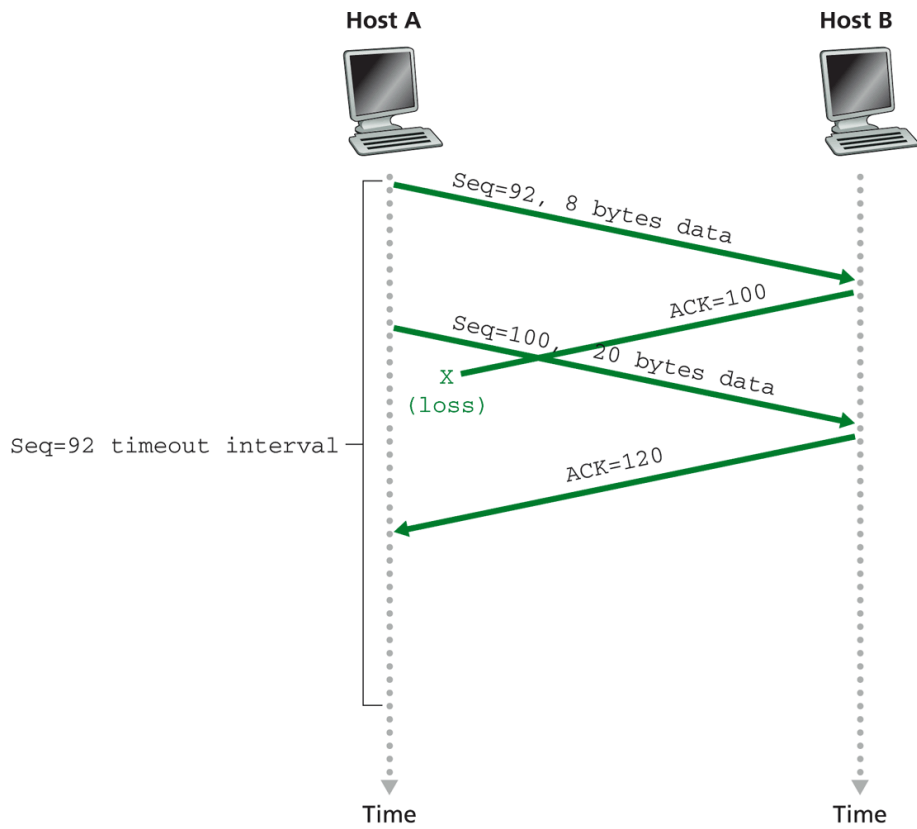
Figure 2.6   A cumulative acknowledgment avoids retransmission of the first segment, from Kurose and Ross [13]

other segments that arrive at the receiver shortly after the segment was lost. Because of cumulative acknowledgements, multiple ACKs are received indicating that the receiver is ready for the byte number associated with the missing packet. These duplicate acknowledgements usually arrive much sooner than the timeout period; therefore, when a duplicate acknowledgement arrives, the first segment that has not been ACKed is sent immediately. TCP actually waits until the third duplicated ACK arrives to assume the segment is lost and thus retransmit this segment. This helps avoid unnecessary transmissions in cases in which segments arrive at the receiver slightly out of order. If a segment is actually lost, a triple duplicate ACK will arrive quickly, since there are usually multiple segments already in transit arriving shortly after the segment is lost. This process is known as fast retransmit and allows for lost segments to be resent much faster than if only timeouts were used.

2-10

Another important feature of TCP is flow control, which ensures the receiver is not overwhelmed and the network is not congested. To ensure that the receiver is not overwhelmed, the sender maintains a variable called the receive window, which is a measure of the available space in the receiver's buffer. The receiver advertises this value in the segments that are sent to the sender. A variable that is maintained by the receiver is the congestion window, which is used to throttle the sender in order prevent congestion in the network. The send window corresponds to the number of unacknowledged packets allowed in the network, and is calculated by taking the minimum of the receive and congestion windows. In Figure 2.2, part A has a send window of 1, while part B has a send window of 3. It can be seen that as the send window increases, the overall send rate increases.

To understand how TCP handles congestion control, assume that the receive window is large enough to be ignored and that the number of unacknowledged packets in the network equals the congestion window. In this case, the congestion window also corresponds to the number of packets sent during each round trip time (RTT). The TCP congestion control algorithm has three main components which are discussed below: additive-increase multiplicative-decrease, slow start, and reaction to timeout events.

When networks become congested, they begin to lose packets due to overflowing queues. Therefore, when a loss event is detected possibly indicating congestion, the congestion window is lowered. When a triple duplicate ACK arrives indicating congestion, the congestion window is cut in half, which reduces the sending rate by a factor of two. When segment loss is not being detected, the congestion window is increased by one segment every RTT. This process is known as additive-increase, multiplicative-decrease. This process is shown in Figure 2.7. This saw-tooth pattern is representative of a TCP connection during normal operation. The sending rate is linearly increased until a loss event happens, at which point the sending rate is cut in half. The process starts all over again and the sending rate is repeatedly increased until a loss event happens, followed by a reduction in the congestion window.

When a connection begins, TCP enters the phase known as slow-start. This phase of TCP is called slow-start because the sender begins sending at a slow rate; however, the choice of terminology may seem to be a misnomer, since the goal of slow-start is to begin
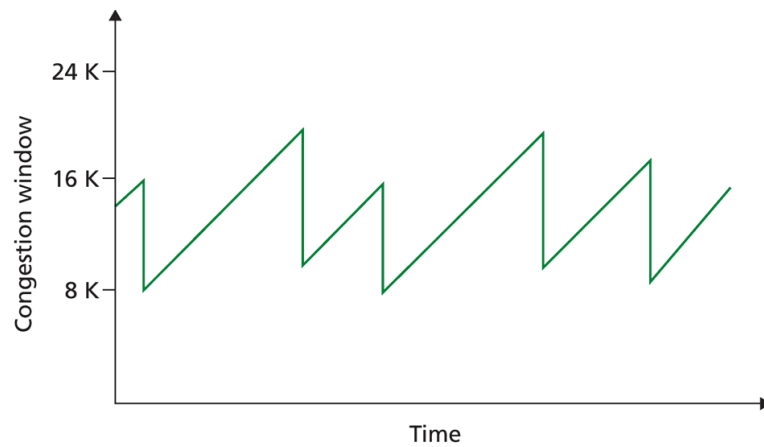
Figure 2.7   Additive-increase, multiplicative-decrease congestion control, from Kurose and Ross [13]

transmitting as fast as possible. Because it would take a long time for the sending rate to reach an acceptable level if the sending rate were increased linearly, the TCP connection begins by doubling its congestion window every RTT, resulting in an exponential increase. The connection window exponentially increases until either a threshold is reached at which point the TCP connection enters additive-increase, multiplicative-decrease, or until a loss event is detected at which point the congestion window is cut in half and then additive-increase, multiplicative-decrease takes over. This process is called slow-start since the sender begins sending at a slow rate.

An early version of TCP called Tahoe [13] enters slow-start whenever a loss event is indicated either by a timeout or duplicate ACK. After either event, the congestion window is set to one and slow-start begins. The threshold for slow-start to end is set to half of the previous congestion window. As discussed earlier, duplicate ACKs usually indicate when a loss event occurs. A timeout is considered a much more serious event because, for this to occur, all segments sent during a timeout period must have been lost. Since duplicate ACKs are not as serious as timeouts, the approach taken by TCP Tahoe was found to be too conservative. The newer version of TCP called Reno [13] only enters slow-start after a time-out. When a triple duplicated ACK is detected, TCP Reno halves the congestion window resulting in the saw-tooth pattern described above. Figure 2.8 shows the behavior of both versions of TCP. Notice a triple duplicate ACK is detected at round 8. The less
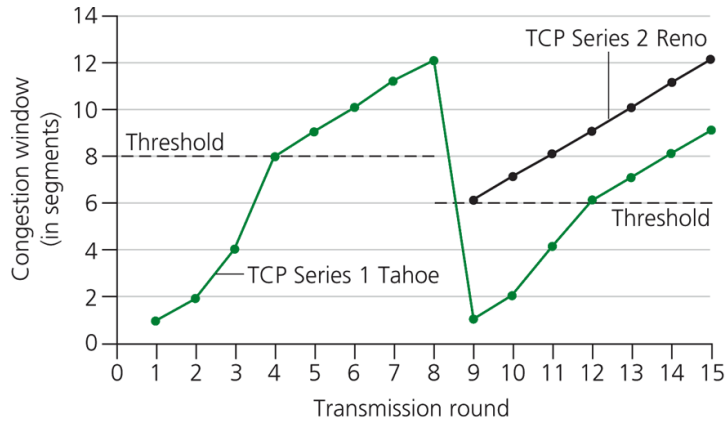
Figure 2.8   Evolution of TCP's congestion window (Tahoe and Reno), from Kurose and Ross [13]

conservative approach taken by TCP Reno is called fast recovery. Versions of TCP based on Reno are the primary versions of TCP in use today.

2.4   Congestion Control with Lossy Data Links

2.4.1   Congestion Control Issue.   The congestion control algorithm used by TCP Tahoe and TCP Reno assumes that the network links are very reliable and only lose data in rare circumstances. These algorithms attribute all packet loss to congestion. This assumption is no longer valid in wireless networks because packet loss is often due to bit-error-rates in the wireless links. When packets are lost due to an error in the link, either timeouts or duplicate ACKs occur just as if the packets were lost due to congestion. In response, the congestion window is lowered, thus reducing the sending rate. Since there is no congestion, this is not the correct action for TCP to take and the efficiency of the TCP connection is greatly reduced. There have been a number of efforts taken to create a congestion control algorithm for TCP that can differentiate between congestion and lossy data links.

2.4.2   TCP Modifications.   TCP Vegas [5] is a modification to TCP Reno that implements a different congestion control algorithm designed to prevent congestion rather than respond to it. TCP Vegas is based on the premise that, as the congestion window is

increased, the sending rate also increases until the network begins to experience congestion. Once congestion begins to exist, the sending rate begins to flatten out, even though the congestion window is still increasing. At this point, the maximum throughput of the system has been reached, and any additional packets sent are buffered by the bottleneck router. It is possible to calculate the expected sending rate that would result if there were no congestion present on the network by:

$$Expected = WindowSize/BaseRTT$$

where $WindowSize$ is the current congestion window size and $BaseRTT$ is the round trip time associated with no congestion. The expected sending rate is compared to the actual sending rate and the difference is taken such that

$$Diff = Expected - Actual$$

Two thresholds are chosen, $\alpha < \beta$, which correspond to having too little and too much extra data in the network, respectively. When $Diff < \alpha$, the congestion window is increased because the number of packets in transit is too small and the network is not being used to its full capacity. Alternately, when $Diff > \beta$, the congestion window is decreased because too much data is in transit, indicating that the network is being congested. The congestion window is left alone when $\alpha \leq Diff \leq \beta$. The authors state that the value for α should correspond to one extra packet in the network and β should correspond to three extra packets in the network. Also the slow-start mechanism is modified in TCP Vegas. In TCP Vegas, the congestion window experiences exponential growth every other RTT. In the period between, the congestion window is kept constant in order for the congestion detection to take place. The authors claim to achieve between 37% - 71% improvement in throughput when compared to TCP Reno. This method of congestion detection improves upon TCP Reno in the wireless setting, but is still not perfect. If packets are lost due to lossy links, the actual sending rate is decreased, which in turn lowers the congestion window. While the decrease in congestion window is not as dramatic as with TCP Reno, it is still not the desired result. Also, TCP Vegas has been shown to have issues when the

return path is congested [10]. Since the ACKs for the sent data are delayed, the actual sending rate is delayed, which also reduces the congestion window unnecessarily.

Another version of TCP called Westwood [7, 15] estimates bandwidth in order to avoid the wireless congestion control issue presented above. TCP Westwood estimates bandwidth by observing the rate in which ACKs are received. Since an ACK is sent when a packet arrives, the arrival of an ACK indicates that a packet was successfully transmitted to the receiver. The bandwidth is calculated by

$$b_k = d_k/\Delta_k$$

where d_k is the size of the packet that was ACKed and Δ_k is the time since the last ACK. In order to obtain an accurate estimate of bandwidth, the bandwidth measurements are low-pass filtered, removing high-frequency noise. TCP Westwood functions similarly to TCP Reno except when a triple duplicated ACK is detected: instead of cutting the congestion window in half, it is set to the bandwidth estimate. Also, when a timeout occurs, the slow start threshold is set to the bandwidth estimate as well, instead of half of the current congestion window. TCP Westwood still increases linearly in order to "search" for extra bandwidth. The authors claim that up to a 550% improvement has been observed over TCP Reno. This benefit is due to the fact that TCP Westwood is able to make an informed decision on how much the congestion window should be reduced, unlike TCP Reno which always reduces the congestion window in half. TCP Westwood was also found to outperform TCP Vegas in all cases and outperformed TCP Vegas by more than a factor of 3 in the case in which the return path was congested [10].

Explicit Congestion Notification (ECN) is a mechanism to inform TCP of congestion [9]. When routers begin to become congested, they set the ECN field in the packets that travel through them. The decision to set the ECN flag is based on a threshold queue size value that is less than the total queue size. This allows the sender to be notified of congestion before loss events occur, since the ECN flag is indicating when congestion is present. In the simplest case, the ECN is treated as a triple duplicate ACK and can be added to any of the TCP approaches above. TCP Jersey [14, 26] combines bandwidth

estimation with ECN. TCP Jersey estimates the bandwidth based on the rate in which ACKs are received in a very similar manner to TCP Westwood. TCP Jersey also uses the bandwidth estimate to set the congestion window. However, when a triple duplicate ACK is detected, TCP Jersey checks to see if the ECN flag is set. If the flag is set, indicating congestion, than the congestion window is reduced to the bandwidth estimate just like TCP Westwood. If the flag is not set, the congestion window is not changed, since the absence of the ECN flag indicates the segments were lost due to link failure. The added information provided by the ECN flag allows TCP Jersey to outperform TCP Westwood by 17% in the scenario presented in [26].

2.5   Summary

In this chapter, an issue with TCP's congestion control has been raised and a number of approaches to solve this issue were described. The Kalman filter was presented as an alternative to traditional network state inference techniques and a few examples of the use of Kalman filters within the network tomography field were given. In the following chapter, a Kalman filter based network state estimator and network queue controller are developed as an alternate solution to the TCP congestion control problem.

III. Methodology

3.1 Introduction

In this chapter, a stochastic control scheme is developed to control the sending rate to a network queue based on the queue size. This controller is developed as an alternative to the flow control mechanisms described in the previous chapter. A system model describing the transient behavior of a network queue is developed, which is needed to formulate both the network state estimator and the network queue controller. The theory and design are presented for both the Kalman filter based network state estimator and the network queue controller. The Kalman filter estimates both the size of a network queue and the packet arrival rate to the queue based on sampled queue size data. These estimates are fed to the network queue controller, which uses the estimates to calculate the packet arrival needed to maintain a desired queue size. The network queue controller is based on a Linear Quadratic Gaussian (LQG) design.

3.2 Model Development

In order to develop the network control scheme described above, a dynamics model describing the transient behavior of a queue is required. The queue is assumed to have the Markov property, which allows the entire state description to be described by only the current state and is also called the *memoryless* property [11]. In the case of a Markovian or memoryless queue, the system state can be completely described by the current size of the queue. The future behavior of the queue does not depend on the past, but only on the current queue size and future inputs. The queue can be modeled as a continuous–parameter Markov chain, which represents the state transition model of a Markovian system [20].

Starting with the Markov chain model for the queue, the equations describing the transient behavior of the queue can be derived. A summary of the derivation presented by Gross and Harris [11], which is an outline of the solution derived by Bailey in 1954 [3], is presented below. The Chapman-Kolmogorov equation for a continuous-parameter Markov chain is given by

$$p_{ij}(s,u) = \sum_r p_{ir}(s,t)p_{rj}(t,u) \tag{3.1}$$

where $p_{ij}(s, u)$ is the probability of transitioning from state i to state j in the time beginning at s and ending at u. In Eq. (3.1), the transition from state i to state j can occur through any other intermediate state r at time t. The choice of the intermediate time t is arbitrary and holds true for all $s < t < u$. Parameters q_i and q_{ij} will be defined as

$$q_i(t)\Delta t + o(\Delta t) = 1 - p_{ii}(t, t + \Delta t) = prob[\text{change of state}(t, t + \Delta t)] \qquad (3.2)$$

$$q_{ij}(t)\Delta t + o(\Delta t) = p_{ij}(t, t + \Delta t) \qquad (3.3)$$

where $o(\Delta t)$ represents the higher order terms. To first order, $q_i(t)$ is the rate in which the state changes from its current state and $q_{ij}(t)$ is the rate at which the state changes from state i to state j.

Given Eq. (3.1) and the definitions in Eq. (3.2) and Eq. (3.3), the forward Kolmogorov equation is found in [11] to be

$$\frac{\partial}{\partial t}p_{ij}(s, t) = -q_j(t)p_{ij}(s, t) + \sum_{r \neq j} p_{ir}(s, t)q_{rj}(t) \qquad (3.4)$$

The forward Kolmogorov equation gives the rate of change of the probability associated with going from state i to state j. By time t, the system can reach any state with some associated probability. In the above equation, the summation term corresponds to the weighted sum of the rates associated with transitioning from this intermediate state to state j. The first term corresponds to the rate at which the state leaves state j given the probability that the system has reached this state at time t. By assuming that the process starts at $s = 0$ and is also homogeneous so that $q_i(t) = q_i$ and $q_{ij}(t) = q_{ij}$, Eq. (3.4) can be written as

$$\frac{dp_{ij}(0, t)}{dt} = -q_i p_{ij}(0, t) + \sum_{r \neq j} p_{ir}(0, t)q_{rj} \qquad (3.5)$$

Multiplying both sides by $p_i(0)$ and summing over all i gives

$$\frac{dp_j(t)}{dt} = -q_j p_j(t) + \sum_{r \neq j} p_r(t)q_{rj} \qquad (3.6)$$

Equation (3.6) calculates the rate of change of the probability associated with being in state j.

Since the number of packets in a queue must be a integer, during an infinitesimal time period, the state can only change by a value of +1 or -1. This is known as a birth-death process, which results in the following:

$$q_{n,n+1} = \lambda_n \tag{3.7a}$$

$$q_{n,n-1} = \mu_n \qquad (\mu_n \neq 0) \tag{3.7b}$$

$$q_{rj} = 0 \qquad \text{(elsewhere)} \tag{3.7c}$$

$$q_n = \lambda_n + \mu_n \qquad (q_0 = \lambda_0) \tag{3.7d}$$

where λ_n and μ_n traditionally represent the packet arrive rate and packet service rate, respectively, for the integer queue size n. The arrival and service rates λ_n and μ_n are not required to be an integer value. By substituting the above values into Eq. (3.6) and denoting queue size with n instead of j, the following infinite set of differential equations are obtained

$$\frac{dp_n(t)}{dt} = -(\lambda_n + \mu_n)p_n(t)$$
$$+\lambda_{n-1}p_{n-1}(t) + \mu_{n+1}p_{n+1}(t) \quad (n \geq 1) \tag{3.8a}$$

$$\frac{dp_0(t)}{dt} = -\lambda_0 p_0(t) + \mu_1 p_1(t) \qquad (n = 0) \tag{3.8b}$$

By assuming that the arrival rates and service rates of the queue are independent of queue size, we can omit subscripts on λ and μ,. allowing Eq. (3.8) to be written as

$$\frac{dp_n(t)}{dt} = -(\lambda + \mu)p_n(t)$$
$$+\lambda p_{n-1}(t) + \mu p_{n+1}(t) \qquad (n > 0) \tag{3.9a}$$

$$\frac{dp_0(t)}{dt} = -\lambda p_0(t) + \mu p_1(t) \qquad (n = 0) \tag{3.9b}$$

$$n(0) = n_0 \tag{3.9c}$$

By using a combination of probability generating functions, partial differential equations, and Laplace transforms [11], the solution to the infinite set of differential equations given by Eq. (3.9) can be found. Details are given in [11] and result in the following solution:

$$p_n(t) = e^{-(\lambda+\mu)t} \left[\left(\frac{\lambda}{\mu}\right)^{(n-n_0)/2} I_{n-n_0}\left(2t\sqrt{\lambda\mu}\right) \right.$$

$$+ \left(\frac{\lambda}{\mu}\right)^{(n-n_0-1)/2} I_{n+n_0+1}\left(2t\sqrt{\lambda\mu}\right) \qquad (3.10)$$

$$\left. + \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \sum_{j=n+n_0+2}^{\infty} \left(\frac{\lambda}{\mu}\right)^{-j/2} I_j\left(2t\sqrt{\lambda\mu}\right) \right]$$

for all $n \geq 0$, where I is the infinite series for the modified Bessel function of the first kind given by

$$I_n(y) = \sum_{k=0}^{\infty} \frac{(y/2)^{n+2k}}{k!(n+k)!} \qquad (n > -1) \qquad (3.11)$$

Although Eq. (3.10) gives the probability density function (pdf) of queue size, it is very complex and time consuming to calculate. Cantrell was able to simplify the calculation using Marcum's Q-functions [6]. The simplified expression is given by

$$p_n(t) = (1-\rho)\rho^n$$

$$+ \rho^n \left(\rho Q_{n+n_0+2}(\alpha,\beta) - Q_{n+n_0+1}(\alpha,\beta)\right)$$

$$+ \begin{cases} Q_{n-n_0+1}(\beta,\alpha) - Q_{n-n_0}(\beta,\alpha) & n > n_0 \\ Q_1(\alpha,\beta) + Q_1(\beta,\alpha) - 1 & n = n_0 \\ Q_{n_0-n+1}(\alpha,\beta) - Q_{n_0-n}(\alpha,\beta) & n < n_0 \end{cases} \qquad (3.12)$$

where

$$\rho = \frac{\lambda}{\mu}, \alpha = \sqrt{2\rho\mu t}, \beta = \sqrt{2\mu t}$$

and Q is given by

$$Q_m(\alpha,\beta) = \exp\left(-\frac{\alpha^2 + \beta^2}{2}\right) \sum_{k=1-m}^{\infty} \left(\frac{\alpha}{\beta}\right)^k I_k(\alpha,\beta) \qquad (3.13)$$

The expected value of the queue size can be calculated from Eq. (3.12) [6].

3-4

For $n_0 \geq 1$

$$
\begin{aligned}
E\left[n(t)|n_0\right] = {} & \frac{\rho}{1-\rho}\left(1 - Q_{n_0+2}(\alpha, \beta)\right) \\
& - \frac{1}{\rho^{n_0}(1-\rho)}\left(1 - Q_{n_0+2}(\beta, \alpha)\right) + n_0 Q_{n_0+1}(\alpha, \beta) \\
& + \rho\mu t Q_{n_0+2}(\alpha, \beta) - \mu t Q_{n_0}(\alpha, \beta)
\end{aligned}
\tag{3.14}
$$

For $n_0 = 0$

$$
\begin{aligned}
E\left[n(t)|n_0 = 0\right] = {} & \frac{\rho}{1-\rho}(1 - Q_2(\alpha, \beta)) \\
& - \frac{1}{1-\rho}(1 - Q_2(\beta, \alpha)) \\
& + \rho\mu t Q_1(\alpha, \beta) - \mu t\left(1 - Q_2(\beta, \alpha)\right)
\end{aligned}
\tag{3.15}
$$

Equations (3.14) and (3.15) provide the expected value of the number of packets in the queue as a function of time, which describes the transient behavior of the queue. These two equations will be the basis of the system model used by the network state estimator and network queue controller developed below.

3.3   Estimation and Control Theory

3.3.1   *Extended Kalman Filter Theory.*   A Kalman filter can be used to estimate the state of the network accurately. Due to the fact that the network model is a nonlinear discrete model with discrete measurements, a discrete-discrete extended Kalman filter will be used. The discrete-discrete extended Kalman filter equations are given below. In the extended Kalman filter, a linear approximation of the nonlinear dynamics equation is used. The nonlinear dynamics equation is relinearized about the current state estimate. For a derivation of the extended Kalman filter, see [17].

Equations (3.16) and (3.17) describe the discrete-time system model.

$$
\mathbf{x}(t_i) = \boldsymbol{\phi}\left(t_i, t_{i\text{-}1}, \mathbf{x}(t_{i\text{-}1})\right) + \mathbf{w_d}(t_i)
\tag{3.16}
$$

Let n be the number of states and m be the number of measurements. Then, $\mathbf{x}(t_i)$ is an n-dimensional vector describing the state of the system at time t_i. The nonlinear dynamics equation for the transition of the states of the system from time t_{i-1} to t_i is ϕ, which is also an n-dimensional vector. The dynamics noise $\mathbf{w_d}$ represents the unknown system dynamics not included in ϕ and is an n-dimensional function containing discrete-time white Gaussian noise of zero mean and covariance kernel

$$E\left\{\mathbf{w_d}(t_i)\mathbf{w_d^T}(t_j)\right\} = \begin{cases} \mathbf{Q_d} & t_i = t_j \\ \mathbf{0} & t_i \neq t_j \end{cases} \tag{3.17}$$

where $\mathbf{Q_d}$ is an n-by-n matrix representing the covariance of $\mathbf{w_d}$. The discrete-time measurement model is given by Eqs. (3.18) and (3.19); linear measurements are assumed here since that is sufficient for this research:

$$\mathbf{z}(t_i) = \mathbf{H}\mathbf{x}(t_i) + \mathbf{v}(t_i) \tag{3.18}$$

where $\mathbf{z}$ is the m-dimensional measurement vector and $\mathbf{H}$ is the m-by-n measurement matrix. Note that $\mathbf{H}$ is assumed constant for this development, but in general can be time varying. The measurement noise, which represents the uncertainty of the measurement, is an m-dimensional vector containing discrete-time white Gaussian noise of zero mean and covariance kernel

$$E\left\{\mathbf{v}(t_i)\mathbf{v^T}(t_j)\right\} = \begin{cases} \mathbf{R} & t_i = t_j \\ \mathbf{0} & t_i \neq t_j \end{cases} \tag{3.19}$$

where $\mathbf{R}$ is an m-by-m matrix representing the covariance of $\mathbf{v}$. The dynamics noise $\mathbf{w_d}$ and measurement noise $\mathbf{v}$ are reasonably assumed to be independent. The linearized state transition matrix, which is an n-by-n matrix, can be found by

$$\mathbf{\Phi}(t_i, t_{i\text{-}1}) = \left.\frac{\partial\phi}{\partial\mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)} \tag{3.20}$$

where $\hat{\mathbf{x}}(t_{i-1}^+)$ is the state estimate provided by the Kalman filter at time t_{i-1} after the measurement update. From Eq. (3.20), the elements of $\boldsymbol{\Phi}$ are calculated by

$$\Phi_{ij}(t_i, t_{i\text{-}1}) = \left.\frac{\partial \phi_i}{\partial \mathrm{x}_j}\right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)} \tag{3.21}$$

Since ϕ is not continuous, Eq. (3.21) must be approximated by replacing the partial derivative with a two-sided difference equation. For example, $\boldsymbol{\Phi}_{11}$ is calculated by

$$\Phi_{11} = \left.\frac{\partial \phi_1}{\partial \mathrm{x}_1}\right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)}$$

$$\approx \left.\frac{\phi_1 \left[\mathbf{x}+\begin{bmatrix}\Delta\mathrm{x}_1\\0\\\vdots\\0\end{bmatrix}\right] - \phi_1 \left[\mathbf{x}-\begin{bmatrix}\Delta\mathrm{x}_1\\0\\\vdots\\0\end{bmatrix}\right]}{2\Delta\mathrm{x}_1}\right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)} \tag{3.22}$$

where $\Delta\mathrm{x}_1$ is a small perturbation in x_1. The value of $\Delta\mathrm{x}$ must be chosen small enough such that the difference equation will give an accurate approximation to the partial derivative, but not so small that numerical precision difficulties occur.

The Kalman filter consists of two major steps: the time propagation step and the measurement update step. In the following equations, the superscript - denotes before the measurement update, and the superscript + denotes after the measurement update. In the time propagation step, the estimate $\hat{\mathbf{x}}$ and covariance $\mathbf{P}$ are propagated from t_{i-1} to t_i by

$$\hat{\mathbf{x}}(t_i^-) = \phi\left(t_i, t_{i\text{-}1}, \hat{\mathbf{x}}(t_{i\text{-}1}^+)\right) \tag{3.23}$$

$$\mathbf{P}(t_i^-) = \boldsymbol{\Phi}(t_i, t_{i\text{-}1})\mathbf{P}(t_{i\text{-}1}^+)\boldsymbol{\Phi}^{\mathrm{T}}(t_i, t_{i\text{-}1}) + \mathbf{Q_d} \tag{3.24}$$

where $\mathbf{P}$ is an n-by-n matrix. In the update step, the measurement is incorporated into the estimate by

$$\mathbf{K}(t_i) = \mathbf{P}(t_i^-)\mathbf{H}^{\mathrm{T}}\left[\mathbf{H}\mathbf{P}(t_i^-)\mathbf{H}^{\mathrm{T}} + \mathbf{R}\right]^{-1} \tag{3.25}$$

$$\hat{\mathbf{x}}(t_i^+) = \hat{\mathbf{x}}(t_i^-) + \mathbf{K}(t_i)\left[\mathbf{z}(t_i) - \mathbf{H}\hat{\mathbf{x}}(t_i^-)\right] \tag{3.26}$$

$$\mathbf{P}(t_i^+) = \mathbf{P}(t_i^-) - \mathbf{K}(t_i)\mathbf{H}\mathbf{P}(t_i^-) \tag{3.27}$$

where the Kalman filter gain $\mathbf{K}$ is an $n-$by$-m$ matrix.

3.3.2 Nonlinear Stochastic Controller Theory. In order to regulate the network queues, a controller is needed. Since the network model is a discrete-time nonlinear model, a discrete-time nonlinear stochastic controller will be designed. Specifically, a controller will be used to provide flow control in a computer network in lieu of the techniques discussed in Section 2.4.2. The controller will be used to regulate the queue size about a chosen setpoint by controlling the sending rate to the queue.

In the linear case, the certainty equivalence principle states that cascading the optimal estimator with the deterministic optimal controller produces an optimal design [18]. In other words, the Kalman filter and deterministic full-state controller can be designed independently of each other and the result would still be optimal. Therefore, in the linear case, the optimal stochastic controller can be designed with the assumption that $\mathbf{x}(t_i)$ is perfectly known and that there is no dynamic noise $\mathbf{w_d}(t_i)$ or measurement noise $\mathbf{v}(t_i)$. Unfortunately, the certainty equivalence principle does not extend to the nonlinear case. However, in order to design a controller in a feasible manner, simplifying assumptions will be made resulting in a suboptimal controller. The *assumed* certainty equivalence design technique will be used in which the controller will be designed as if the certainty equivalence principle could be extended to the nonlinear case. As long as the estimate of $\mathbf{x}(t_i)$ produced by the Kalman filter is good representation of the actual system state, the assumed certainty equivalence design is sufficient in most cases. For the reasons presented above, a deterministic nonlinear controller will be presented.

Dynamic programming may be used to design an optimal deterministic nonlinear controller; however, due to the complexity required, it is desirable to use a different suboptimal synthesis technique. For this reason, a linear perturbation control law is used in order to exploit the systematic nature of the Liner Quadratic Gaussian (LQG) design. In addition to linearizing the system model, the controller is simplified further by implementing a *steady state constant gain* control law. The selection of *steady state constant gains*

simplify the synthesis of the controller and allow the controller gains to be precomputed, reducing the computational burden.

In order to achieve type-1 deterministic control in which a desired setpoint is maintained with zero steady state error, a proportional-plus-integral (PI) controller is used. In this section, the equations needed for the LQG synthesis of a *steady state* linear perturbation PI controller are presented. For a derivation of these equations, see [18]. The description of the system to be controlled is given by

$$\mathbf{x}(t_{i+1}) = \boldsymbol{\phi}\left(t_{i+1}, t_i, \mathbf{x}(t_i), \mathbf{u}(t_i)\right) \tag{3.28a}$$

$$\mathbf{y_c}(t_i) = \mathbf{c}[\mathbf{x}(t_i), \mathbf{u}(t_i)] \tag{3.28b}$$

Equation (3.28a) is the deterministic version of Eq. (3.16), where $\boldsymbol{\phi}$ also depends on the output of the controller $\mathbf{u}(t_i)$. The controlled variables $\mathbf{y_c}(t_i)$ are related to the states and controller inputs by Eq. (3.28b).

In order use LQG synthesis, the system model must be linearized. It is chosen to linearize about the equilibrium solution, which is found by

$$\mathbf{x}(t_{i+1}) = \mathbf{x}(t_i) = \mathbf{x_0} = \boldsymbol{\phi}\left(t_{i+1}, t_i, \mathbf{x_0}, \mathbf{u_0}\right) \tag{3.29a}$$

$$\mathbf{y_{c0}} = \mathbf{y_d} = \mathbf{c}[\mathbf{x_0}, \mathbf{u_0}] \tag{3.29b}$$

where $\mathbf{y_d}$ is the desired setpoint of the system. The solution is represented as

$$x_0(\mathbf{y_d}) = \mathbf{x_0} \tag{3.30a}$$

$$u_0(\mathbf{y_d}) = \mathbf{u_0} \tag{3.30b}$$

which are functions of the setpoint $\mathbf{y_d}$. Therefore, a different controller must be computed for each setpoint. The linear time-invariant perturbation system model is given by

$$\delta\mathbf{x}(t_{i+1}) = \boldsymbol{\Phi}\delta\mathbf{x}(t_i) + \mathbf{B_d}\delta\mathbf{u}(t_i) \tag{3.31a}$$

$$\delta\mathbf{y_c}(t_i) = \mathbf{C}\delta\mathbf{x}(t_i) + \mathbf{D_y}\delta\mathbf{u}(t_i) \tag{3.31b}$$

where

$$\delta\mathbf{x}(t_i) \;=\; \mathbf{x}(t_i) - \mathbf{x_0}; \qquad \mathbf{x_0} = x_0(\mathbf{y_d}) \tag{3.32}$$

$$\delta\mathbf{u}(t_i) \;=\; \mathbf{u}(t_i) - \mathbf{u_0}; \qquad \mathbf{u_0} = u_0(\mathbf{y_d}) \tag{3.33}$$

$$\delta\mathbf{y_c}(t_i) \;=\; \mathbf{y_c}(t_i) - \mathbf{y_{c0}}; \qquad \mathbf{y_{c0}} = \mathbf{y_d} \tag{3.34}$$

The matrices that define the perturbation system are found by evaluating partials of ϕ and $\mathbf{c}$ at the equilibrium values:

$$
\begin{aligned}
\mathbf{\Phi} &= \left.\frac{\partial\phi}{\partial\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}} & \mathbf{B_d} &= \left.\frac{\partial\phi}{\partial\mathbf{u}}\right|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}} \\
\mathbf{C} &= \left.\frac{\partial\mathbf{c}}{\partial\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}} & \mathbf{D_y} &= \left.\frac{\partial\mathbf{c}}{\partial\mathbf{u}}\right|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}}
\end{aligned}
\tag{3.35}
$$

In order to achieve the type-1 control described above, a pseudointegral state $\delta\mathbf{q}$ is created by summing the regulation error:

$$\delta\mathbf{q}(t_{i+1}) = \delta\mathbf{q}(t_i) + \delta\mathbf{y_c}(t_i) \tag{3.36}$$

This state is augmented to the system model given by Eq. (3.31a) to become

$$
\begin{bmatrix} \delta\mathbf{x}(t_{i+1}) \\ \delta\mathbf{q}(t_{i+1}) \end{bmatrix}
=
\begin{bmatrix} \mathbf{\Phi} & \mathbf{0} \\ \mathbf{C} & \mathbf{I} \end{bmatrix}
\begin{bmatrix} \delta\mathbf{x}(t_i) \\ \delta\mathbf{q}(t_i) \end{bmatrix}
+
\begin{bmatrix} \mathbf{B_d} \\ \mathbf{D_y} \end{bmatrix}
\delta\mathbf{u}(t_i)
\tag{3.37}
$$

The state transition matrix $\mathbf{\Phi}$ and input matrix $\mathbf{B_d}$ of the augmented system are

$$
\mathbf{\Phi}_{Aug} = \begin{bmatrix} \mathbf{\Phi} & \mathbf{0} \\ \mathbf{C} & \mathbf{I} \end{bmatrix}, \quad
\mathbf{B_{d}}_{Aug} = \begin{bmatrix} \mathbf{B_d} \\ \mathbf{D_y} \end{bmatrix}
\tag{3.38}
$$

The optimal controller is found by minimizing the cost function given by

$$
\begin{aligned}
J \;=\; & \sum_{i=0}^{N} \left\{ \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}(t_i) \\ \delta\mathbf{q}(t_i) \\ \delta\mathbf{u}(t_i) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{X}_{11} & \mathbf{X}_{12} & \mathbf{S}_1 \\ \mathbf{X}_{12}^{\mathrm{T}} & \mathbf{X}_{22} & \mathbf{S}_2 \\ \mathbf{S}_1^{\mathrm{T}} & \mathbf{S}_2^{\mathrm{T}} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}(t_i) \\ \delta\mathbf{q}(t_i) \\ \delta\mathbf{u}(t_i) \end{bmatrix} \right\} \\
& + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}(t_{N+1}) \\ \delta\mathbf{q}(t_{N+1}) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{X_{f11}} & \mathbf{X_{f12}} \\ \mathbf{X_{f12}^{\mathrm{T}}} & \mathbf{X_{f22}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}(t_{N+1}) \\ \delta\mathbf{q}(t_{N+1}) \end{bmatrix}
\end{aligned}
\tag{3.39}
$$

where $\mathbf{X}$ is the state weighting matrix, $\mathbf{X_f}$ is the final state weighting matrix, $\mathbf{U}$ is the control weighting matrix, and $\mathbf{S}$ is the cross-term weighting matrix. These weighting matrices are the tuning parameters for the controller and are adjusted until the desired performance is achieved. The incremental form of the *steady state* control law which minimizes Eq. (3.39) is

$$
\begin{aligned}
\mathbf{u}^*(t_i) = \; & \mathbf{u}^*(t_{i-1}) - \bar{\mathbf{G}}_{\mathbf{c1}}^*[\mathbf{x}(t_i) - \mathbf{x}(t_{i-1})] \\
& + \bar{\mathbf{G}}_{\mathbf{c2}}^* \left\{ \mathbf{y_d}(t_{i-1}) - \mathbf{c}\left[\mathbf{x}(t_{i-1}), \mathbf{u}(t_{i-1})\right] \right\} \\
& + \mathbf{E}\left[\mathbf{y_d}(t_i) - \mathbf{y_d}(t_{i-1})\right]
\end{aligned}
\tag{3.40}
$$

where $\bar{\mathbf{G}}_{\mathbf{c}}^* = \begin{bmatrix} \bar{\mathbf{G}}_{\mathbf{c1}}^* & \bar{\mathbf{G}}_{\mathbf{c2}}^* \end{bmatrix}$ is found by solving the discrete-time algebraic Riccati equation given by

$$
\bar{\mathbf{G}}_{\mathbf{c}}^* \;=\; \left[\mathbf{U} + \mathbf{B}_{\mathbf{d}_{Aug}}^{\mathrm{T}} \bar{\mathbf{K}}_{\mathbf{c}} \mathbf{B}_{\mathbf{d}_{Aug}}\right]^{-1} \left[\mathbf{B}_{\mathbf{d}_{Aug}}^{\mathrm{T}} \bar{\mathbf{K}}_{\mathbf{c}} \mathbf{\Phi}_{Aug} + \mathbf{S}^{\mathrm{T}}\right]
\tag{3.41a}
$$

$$
\bar{\mathbf{K}}_{\mathbf{c}} \;=\; \mathbf{X} + \mathbf{\Phi}_{Aug}^{\mathrm{T}} \bar{\mathbf{K}}_{\mathbf{c}} \mathbf{\Phi}_{Aug} - \left[\mathbf{B}_{\mathbf{d}_{Aug}}^{\mathrm{T}} \bar{\mathbf{K}}_{\mathbf{c}} \mathbf{\Phi}_{Aug} + \mathbf{S}^{\mathrm{T}}\right]^{\mathrm{T}} \bar{\mathbf{G}}_{\mathbf{c}}^*
\tag{3.41b}
$$

and $\mathbf{E}$ is found by

$$
\mathbf{E} = \left[\bar{\mathbf{G}}_{\mathbf{c1}}^* - \bar{\mathbf{G}}_{\mathbf{c2}}^* \bar{\mathbf{K}}_{\mathbf{c22}}^{-1} \bar{\mathbf{K}}_{\mathbf{c12}}^{\mathrm{T}}\right] \mathbf{\Pi}_{12} + \mathbf{\Pi}_{22}
\tag{3.42}
$$

where $\mathbf{\Pi}_{12}$ and $\mathbf{\Pi}_{22}$ are found by

$$
\mathbf{x_0} \;=\; \mathbf{\Pi}_{12}\mathbf{y_d}
\tag{3.43a}
$$

$$
\mathbf{u_0} \;=\; \mathbf{\Pi}_{22}\mathbf{y_d}
\tag{3.43b}
$$

3.4   Design of Network Estimator and Controller

3.4.1   *Design of Network State Estimator.*   Using the extended Kalman filter theory and queueing network model developed earlier, a network state estimator can be developed. The network state estimator will estimate the state of the network based on measurement of the queue size. The state vector consists of two states.

$$\mathbf{x}(t_i) = \begin{bmatrix} x_1(t_i) \\ x_2(t_i) \end{bmatrix} \tag{3.44}$$

The first state $x_1(t_i)$ corresponds to the queue size in packets and the second state $x_2(t_i)$ corresponds to the packet arrival rate to the queue, which was denoted as λ earlier. To define the system and measurement models given by Eqs. (3.16)-(3.19), ϕ, $\mathbf{Q_d}$, $\mathbf{H}$, and $\mathbf{R}$ must be found. The nonlinear dynamics equation associated with x_1 is found from Eqs. (3.14) and (3.15) to be

For $\hat{x}_1(t_{i-1}^+) \geq 1$

$$
\begin{aligned}
\phi_1 &= \frac{\rho}{1-\rho}\left(1 - Q_{\hat{x}_1(t_{i-1}^+)+2}(\alpha,\beta)\right) \\
&\quad - \frac{1}{\rho^{\hat{x}_1(t_{i-1}^+)}(1-\rho)}\left(1 - Q_{\hat{x}_1(t_{i-1}^+)+2}(\beta,\alpha)\right) \\
&\quad + \hat{x}_1(t_{i-1}^+)Q_{\hat{x}_1(t_{i-1}^+)+1}(\alpha,\beta) \\
&\quad + \rho\mu\Delta t Q_{\hat{x}_1(t_{i-1}^+)+2}(\alpha,\beta) - \mu\Delta t Q_{\hat{x}_1(t_{i-1}^+)}(\alpha,\beta)
\end{aligned}
\tag{3.45}
$$

For $\hat{x}_1(t_{i-1}^+) = 0$

$$
\begin{aligned}
\phi_1 &= \frac{\rho}{1-\rho}(1 - Q_2(\alpha,\beta)) \\
&\quad - \frac{1}{1-\rho}(1 - Q_2(\beta,\alpha)) \\
&\quad + \rho\mu\Delta t Q_1(\alpha,\beta) - \mu\Delta t (1 - Q_2(\beta,\alpha))
\end{aligned}
\tag{3.46}
$$

where

$$\Delta t = t_i - t_{i-1}, \; \rho = \frac{\hat{x}_2(t_{i-1}^+)}{\mu}, \; \alpha = \sqrt{2\rho\mu\Delta t}, \; \beta = \sqrt{2\mu\Delta t} \tag{3.47}$$

Due to the unknown nature of the packet arrival rate, the dynamics equation associated with the second state is approximated as a Brownian motion process, which results in

$$\phi_2 = \hat{x}_2(t_{i-1}^+) \tag{3.48}$$

The covariance of the dynamic noise vector $\mathbf{w_d}$ is given by

$$\mathbf{Q_d} = \begin{bmatrix} Q_{d_{11}} & 0 \\ 0 & Q_{d_{22}} \end{bmatrix} \tag{3.49}$$

where the variances of the dynamic noises associated with x_1 and x_2 are given by $Q_{d_{11}}$ and $Q_{d_{22}}$ respectively. The two dynamic noises are treated as uncorrelated which results in a diagonal $\mathbf{Q_d}$. $Q_{d_{11}}$ and $Q_{d_{22}}$ can be thought of as tuning parameters for the filter. Their values will be chosen in an iterative nature such that adequate performance is achieved by the filter. Since only measurements of the first state will be taken, the measurement matrix is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.50}$$

The measurement noise variance R is a scalar value and will be determined by the amount of uncertainty contained in the queue size measurement. The elements of $\mathbf{\Phi}$ are calculated as in Eq. (3.22):

$$\Phi_{11} = \left. \frac{\phi_1\left[\mathbf{x}+\begin{bmatrix}1\\0\end{bmatrix}\right] - \phi_1\left[\mathbf{x}-\begin{bmatrix}1\\0\end{bmatrix}\right]}{2} \right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)} \tag{3.51a}$$

$$\Phi_{12} = \left. \frac{\phi_1\left[\mathbf{x}+\begin{bmatrix}0\\\frac{x_2}{100}\end{bmatrix}\right] - \phi_1\left[\mathbf{x}-\begin{bmatrix}0\\\frac{x_2}{100}\end{bmatrix}\right]}{2\frac{x_2}{100}} \right|_{\mathbf{x}=\hat{\mathbf{x}}(t_{i-1}^+)} \tag{3.51b}$$

$$\Phi_{21} = 0, \quad \Phi_{22} = 1 \tag{3.51c}$$

In the calculation of Φ_{11}, Δx_1 is chosen to be 1, since it is the smallest perturbation allowed due to the fact that the queue size must be an integer. However, there is no such integer requirement for the second state variable; therefore, in the calculation of Φ_{12},

$\Delta\mathrm{x}_2$ was chosen to be $\frac{\mathrm{x}_2}{100}$ in order to provide a scaled perturbation that was two orders of magnitude smaller than the state. All of the necessary components of the Extended Kalman Filter equations given by Eqs. (3.23)-(3.27) have now been defined.

3.4.2 Design of Network Queue Controller. A controller can be designed based on the nonlinear stochastic controller theory presented above. The state vector is the same as Eq. (3.44); and its estimate $\hat{\mathbf{x}}(t_i)$ is provided by the extended Kalman filter designed above. The control input $\mathrm{u}(t_i)$ is the sending rate from the source. The dynamics model in Eq. (3.28a) is identical to Eq. (3.45) and Eq. (3.46), and the change in control input $\Delta\mathrm{u}(t_i)$ is added to ρ in Eq. (3.47).

For $\hat{\mathrm{x}}_1(t_{i-1}^+) \geq 1$

$$
\begin{aligned}
\phi_1 = {} & \frac{\rho}{1-\rho}\left(1 - Q_{\hat{\mathrm{x}}_1(t_{i-1}^+)+2}(\alpha,\beta)\right) \\
& - \frac{1}{\rho^{\hat{\mathrm{x}}_1(t_{i-1}^+)}(1-\rho)}\left(1 - Q_{\hat{\mathrm{x}}_1(t_{i-1}^+)+2}(\beta,\alpha)\right) \\
& + \hat{\mathrm{x}}_1(t_{i-1}^+)Q_{\hat{\mathrm{x}}_1(t_{i-1}^+)+1}(\alpha,\beta) \\
& + \rho\mu\Delta t Q_{\hat{\mathrm{x}}_1(t_{i-1}^+)+2}(\alpha,\beta) - \mu\Delta t Q_{\hat{\mathrm{x}}_1(t_{i-1}^+)}(\alpha,\beta)
\end{aligned}
\tag{3.52}
$$

For $\hat{\mathrm{x}}_1(t_{i-1}^+) = 0$

$$
\begin{aligned}
\phi_1 = {} & \frac{\rho}{1-\rho}(1 - Q_2(\alpha,\beta)) \\
& - \frac{1}{1-\rho}\left(1 - Q_2(\beta,\alpha)\right) \\
& + \rho\mu\Delta t Q_1(\alpha,\beta) - \mu\Delta t\left(1 - Q_2(\beta,\alpha)\right)
\end{aligned}
\tag{3.53}
$$

where
$$
\Delta t = t_i - t_{i-1},\ \rho = \frac{\hat{\mathrm{x}}_2(t_{i-1}^+) + \Delta\mathrm{u}(t_i)}{\mu},\ \alpha = \sqrt{2\rho\mu\Delta t},\ \beta = \sqrt{2\mu\Delta t}
\tag{3.54}
$$

The controlled variable from Eq. (3.28b) is

$$
\mathrm{y}_c(t_i) = \mathrm{x}_1(t_i)
\tag{3.55}
$$

Equation (3.40) will be implemented in the actual controller as

$$u^*(t_i) = u^*(t_{i-1}) + \Delta u^*(t_i) \tag{3.56a}$$

$$\Delta u^*(t_i) = \Delta u^*(t_{i-1}) - \bar{\mathbf{G}}^*_{\mathbf{c1}}[\hat{\mathbf{x}}(t_i) - \hat{\mathbf{x}}(t_{i-1})]$$

$$+ \bar{G}^*_{c2}\{y_d(t_{i-1}) - \hat{x}_1(t_i)\}$$

$$+ E[y_d(t_i) - y_d(t_{i-1})] \tag{3.56b}$$

where $y_d(t_i)$ is the setpoint of the queue in number of packets and the constant gains, $\bar{\mathbf{G}}^*_{\mathbf{c}} = \begin{bmatrix} \bar{\mathbf{G}}^*_{\mathbf{c1}} & \bar{G}^*_{c2} \end{bmatrix}$ and E will be computed offline before controller operation, where $\bar{\mathbf{G}}^*_{\mathbf{c}}$ is a 1-by-3 matrix, $\bar{\mathbf{G}}^*_{\mathbf{c1}}$ is a 1-by-2 matrix, and $\bar{G}^*_{c2}$ is a 1-by-1 matrix in this case. To find the gains, the equilibrium solution is found by Eqs. (3.29a) and (3.29b) to be

$$\Delta u_0 = 0 \tag{3.57a}$$

$$x_0 = \begin{bmatrix} y_d \\ \frac{y_d \mu}{1 + y_d} \end{bmatrix} \tag{3.57b}$$

$\boldsymbol{\Phi}$ is then found to be

$$\Phi_{11} = \left. \frac{\phi_1 \left[\mathbf{x} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right] - \phi_1 \left[\mathbf{x} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right]}{2} \right|_{\mathbf{x}=\mathbf{x}_0, \Delta u_0 = 0} \tag{3.58a}$$

$$\Phi_{12} = \left. \frac{\phi_1 \left[\mathbf{x} + \begin{bmatrix} 0 \\ \frac{x_2}{100} \end{bmatrix} \right] - \phi_1 \left[\mathbf{x} - \begin{bmatrix} 0 \\ \frac{x_2}{100} \end{bmatrix} \right]}{2\frac{x_2}{100}} \right|_{\mathbf{x}=\mathbf{x}_0, \Delta u_0 = 0} \tag{3.58b}$$

$$\Phi_{21} = 0, \Phi_{22} = 1 \tag{3.58c}$$

Next, $\mathbf{B_d}$ is found to be

$$B_{d_1} = \left. \frac{\phi_1 \left[\mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{100} \end{bmatrix} \right] - \phi_1 \left[\mathbf{x} - \begin{bmatrix} 0 \\ \frac{1}{100} \end{bmatrix} \right]}{\frac{1}{50}} \right|_{\mathbf{x}=\mathbf{x}_0, \Delta u_0 = 0} \tag{3.59a}$$

$$B_{d_2} = 1 \tag{3.59b}$$

and $\mathbf{C}$ and $\mathbf{D_y}$ are found to be

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.60}$$

$$D_y = 0; \tag{3.61}$$

The weighting matrix $\mathbf{X}$ is chosen to be diagonal and is structured as

$$\mathbf{X} = \begin{bmatrix} X_{11} & 0 & 0 \\ 0 & X_{22} & 0 \\ 0 & 0 & X_{33} \end{bmatrix} \tag{3.62}$$

and $\mathbf{S}$ is chosen to be

$$\mathbf{S} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.63}$$

where X_{11}, X_{22}, X_{33}, and U are tuning parameters for the controller and are chosen in an iterative manner until the desired performance is obtained. The weights X_{11} and X_{22} correspond to the queue size and queue arrival rate respectively. As these values are increased, the controller will place more importance to regulating these values toward the nominal value. The tuning parameter X_{33} corresponds to the pseudointegral state $\delta\mathbf{q}$. As X_{33} is increased, the importance of minimizing regulation error is increased. Increasing X_{33} will increase the bandwidth of the system. The weight U corresponds to the importance of minimizing control action. Increasing U will result in more liberal control inputs. $\bar{\mathbf{G}}_{\mathbf{c}}^{*}$ and $\bar{\mathbf{K}}_{\mathbf{c}}$ can now be found by solving the Riccati equation given by Eq. (3.41). Finally, E can be found by Eqs. (3.42) and (3.43). All components of Eq. (3.40) have now been defined, thus completing the controller design.

3.5   Summary

In this chapter, a transient queue model was developed. The general extended Kalman filter and an LQG steady state linear perturbation PI controller equations were presented. Based on these equations and the transient queue model, a network state estimator and network queue controller were designed. The performance of these designs are explored in the following chapter.

IV. Results

4.1 Model Validation

This chapter presents the validation of the transient queue model developed in Section 3.2. The model is verified under ideal conditions that exactly match the input traffic assumptions made during the development of the model. The network queueing model is then verified by a more detailed network simulation that involves complex models of network hardware operating under ideal traffic situations. More realistic traffic is simulated to validate the queue model further. The performance of the two-state extended Kalman filter network estimator developed in Section 3.4.1 is demonstrated and the LQG *steady state* linear perturbation PI controller developed in Section 3.4.2 is shown regulating queue sizes.

4.1.1 Simple Queue Results. The analytical model given by Equations (3.14) and (3.15) were modeled in MATLAB, version R2006a [25], and the provided Q function solver was used to calculate the Q values. To validate the transient queue model , a utilization of .85 was chosen to load the queue sufficiently in order to provide a reasonable dynamic response. A sending rate of 5 packets/second was arbitrarily chosen as a reasonable sending rate, resulting in a service rate of 5.88 packets/second. Figure 4.1 displays the results obtained for a utilization $\rho = .85$ and a service rate of $\mu = 5.88$ packets/second. As a check, the steady state result can be compared to the steady state queue size equation given by [20]. The steady state value of 5.67 matches the value obtained from (4.1): $.85/(1 - .85) = 5.67$.

$$n_{steady\ state} = \frac{\rho}{1 - \rho} \tag{4.1}$$

To validate the analytical model, the network simulation tool OPNET modeler version 11.5 [19] was used to simulate a queuing network. First, a simple queuing model was simulated to validate the analytical model under an ideal case. The simple OPNET queuing model is shown in Fig 4.2. The standard OPNET source, queue, and sink models are used to create the simple queueing model. This simple model consists of a simple source generating Poisson traffic with exponentially distributed packet sizes, which is then deliv-
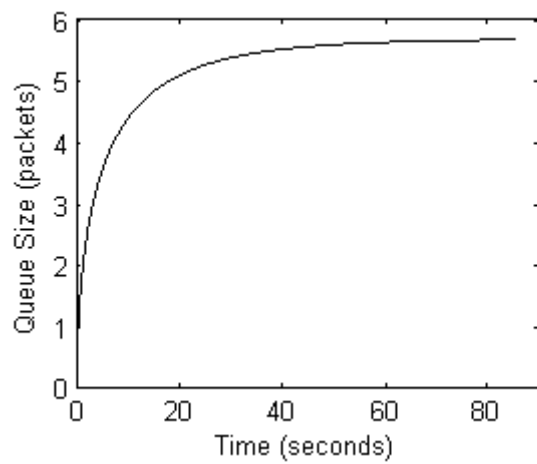
Figure 4.1   The theoretical expected queue size is shown for a utilization of .85 and a service rate of 5.88 *packets*/second
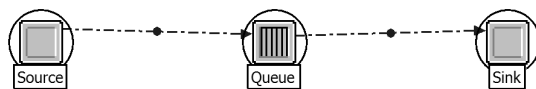

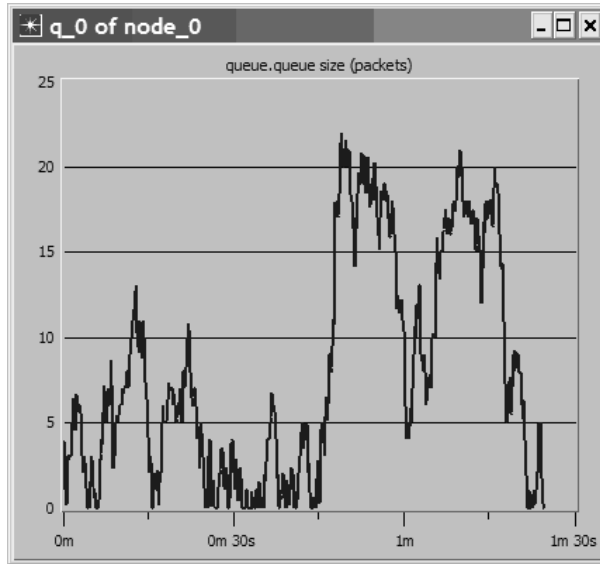Figure 4.2   OPNET model of the simple queuing system.

Figure 4.3   The number of *packets* contained in the queue is shown for a single simulation
              run of the simple queue model with a utilization of .85 and service rate of
              5.88 *packets*/second.

ered to the queue. The queue services each bit of the received traffic at a constant rate,
which due to the exponentially distributed packet sizes, results in an exponential service
rate. The packets are then delivered to the sink. The arrival rate was set arbitrarily to 5
packets/second,corresponding to a service rate of 5.88 packets/second. The mean packet
size is 1500 bytes. The service rate of the queue was set to 70,588 bits/second in order to
match the service rate of 5.88 packets/second with a utilization of .85 as shown in Figure
4.1.

Figure 4.3 displays the results obtained from a single run of the simple queue simu-
lation. The burstiness of the Poisson traffic has a very apparent effect on the queue size
and the results can vary greatly for different runs using different random seed values to
generate the Poisson traffic. To compare the results obtained to the expected value results
shown in Figure 4.1 properly, a Monte Carlo analysis should be conducted using multiple
seed values. To compare typical queue behavior to the theoretical calculations, an eleven
run Monte Carlo analysis was performed. The mean and one standard deviation confidence
bounds of the queue size are calculated from the eleven runs and are shown in Figure 4.4.
Due to the reasons above, all future queue size results shown in this section will be the
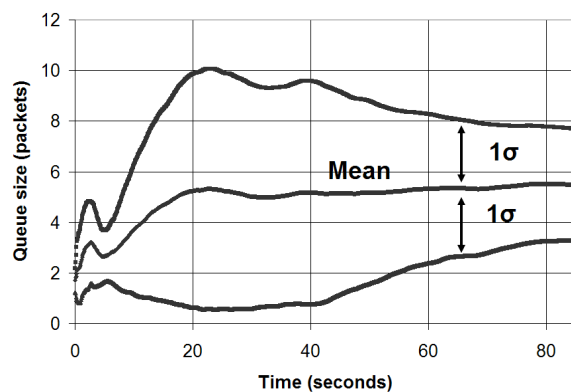
4-3

Figure 4.4   Eleven-run Monte Carlo results obtained from the simple queue model with a utilization of .85 and a service rate of 5.88 *packets*/second.

result of the described Monte Carlo analysis. Notice that the mean value matches very closely with the theoretical results shown in Figure 4.1. Also, notice the wide spread in the confidence bounds due to the large variance in the transient response from run to run.

Due to this wide variance in transient queue behavior, individual results will be impossible to predict precisely; however, the average of many runs will closely match the calculated expected value. The results obtained in this section validate that, under ideal conditions, the expected value equations obtained earlier are correct.

4.1.2   Detailed Network Model with Ideal Traffic.   In the previous section, the transient queue model was validated under ideal circumstances. In this section, a more accurate computer network model is built in OPNET to verify that the transient queue model is valid in an actual network. The detailed network model can be seen in Figure 4.5. To observe the effect of cascading multiple routers, the network was modeled with two workstations connected by two routers. Duplex point-to-point links are used to connect the nodes. The standard OPNET models are used for the workstations and routers, and the routers are modeled with a central processor. Data from the incoming links are sent to a single queue located at the front end of the processor. The processed packets are then sent to the outgoing links. Each outgoing link then contains a queue. The Routing Information Protocol (RIP) is used to route traffic.
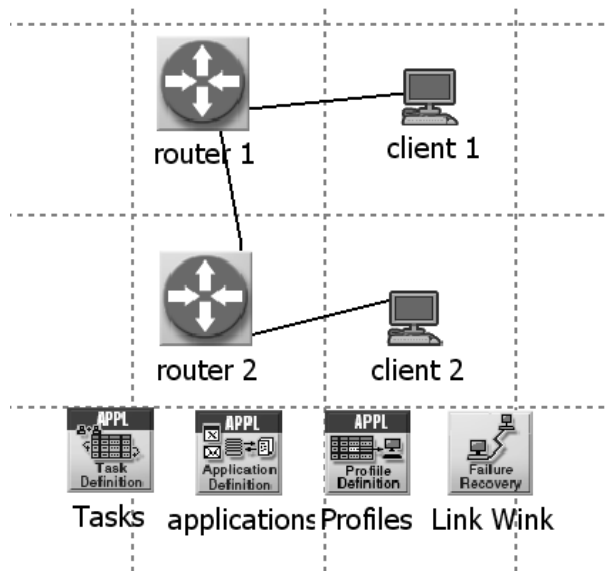
Figure 4.5   OPNET model of the detailed queuing network.

A custom application was modeled in OPNET to recreate the traffic flow from the previous section. The Tasks, Applications, and Profiles blocks in Figure 4.5 were used to model this custom application. The Link Wink block was not used in this experiment. The traffic was sent from client 1 to client 2 using the UDP transport protocol. As in the previous case, Poisson traffic was sent with a mean arrival rate of 5 packets/second and a mean packet size of 1500 bytes. As can be seen in Figure 4.6, the custom application sends 5 packets/second as expected; however, router 1 receives between 6 and 8 packets/second. This can be explained by the presence of small control packets being used by the network. Small control packets are used in the network to perform various tasks including routing and transmission control. Notice that in Figure 4.7, the volume of traffic received by router 1 in bytes is virtually identical to the traffic sent by the application. While the small control packets contribute to the overall number in the stream, due to their small size, they do not contribute much to the overall volume sent in bytes.

The service rates of the routers are set such that the CPU's are 85% utilized, corresponding to the theoretical calculation shown before in Figure 4.1. The links are set to a capacity of 20 Mbits/sec, which is sufficiently high so that the only bottlenecks are the central processor and corresponding queue in each router.
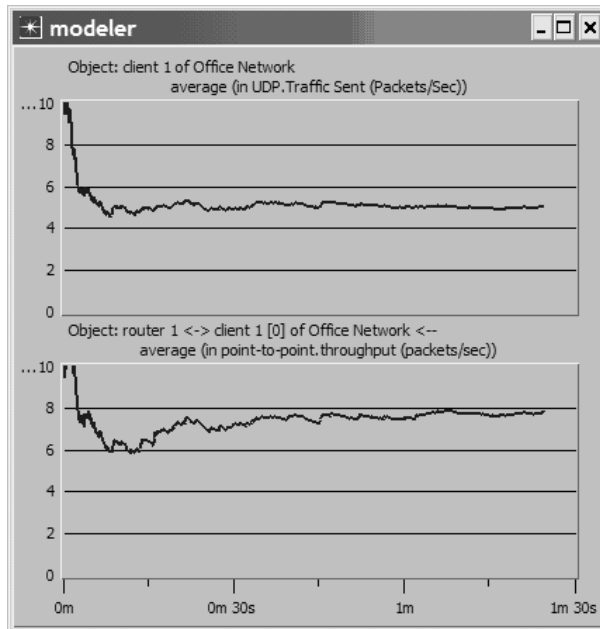
Figure 4.6   The top graph displays the average number of *packets* sent by the custom application and the bottom graph displays the number of *packets* received by router 1.
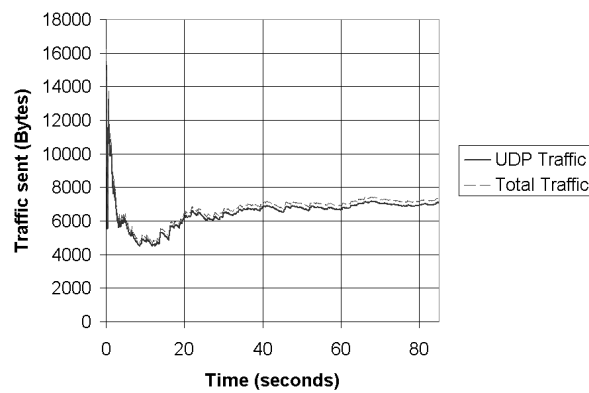

Figure 4.7   Comparison between traffic sent by the application and total traffic received by router 1.
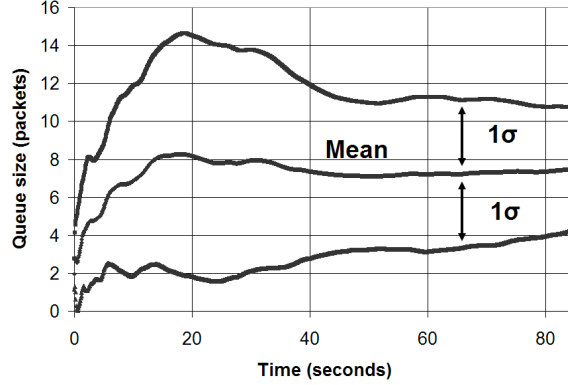
Figure 4.8   Router 1 central queue size in *packets*, 85% loaded, eleven-run Monte Carlo analysis.
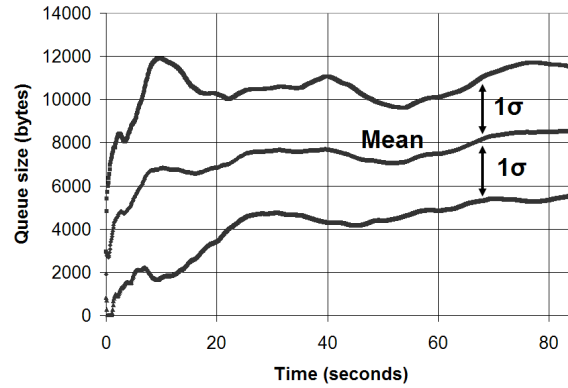

Figure 4.9   Router 1 central queue size in *bytes*, 85% loaded, eleven-run Monte Carlo analysis.

The transient behavior of the central queue in router 1 is displayed in Figure 4.8. Notice that the transient behavior of the queue resembles the calculated value in Figure 4.1 with the exception that it is shifted up by a couple of packets due to the presence of the small control packets discussed earlier. Since the control packets are small, they do not affect the loading of the queue very much and as a result do not affect the overall transient behavior of the queue. However, while in the queue, the control packets are stuck between the larger data packets, which increases the observed value by a few packets. One would still expect the queue size in bytes to be dominated by the application traffic. If this is the case, then the steady state value of the queue size in bytes would be close to 5.67*1500 = 8505 bytes, which can be seen in Figure 4.9.
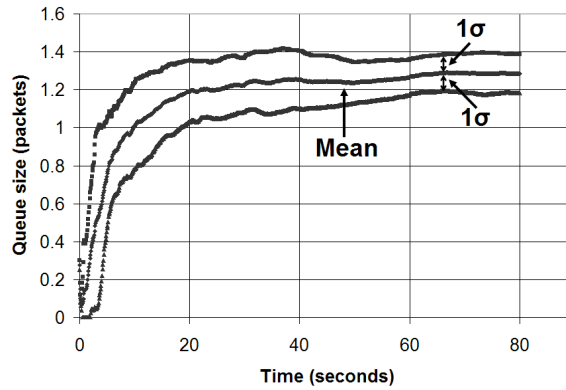
Figure 4.10   Router 2 central queue size in *packets*, 85% loaded, eleven-run Monte Carlo analysis.

The transient response of the queue in the second router can be seen in Figure 4.10. The behavior of the queue does not match the theoretical results obtained in Figure 4.1. In the case of tandem queues, the Markovian nature of the system begins to fall apart. The service rates of the queues are dependant on the packet sizes. Since computer networks typically have a minimum packet size, the minimum allowable service time is non-zero and the service rates are only approximately exponential. When the arrivals to a queue are Poisson, the queue model developed still provides a good approximation of queue size even though the service rate is not completely exponential. However, the presence of a minimum allowable service time reduces the burstiness of the outgoing traffic and the inter-arrival time to the second queue is no longer strictly Poisson. This results in too many assumptions being invalidated at the second queue and the model developed is no longer adequate. In most cases, the output of the first queue will be mixed with other traffic before arriving at the second queue, which restores the Poisson nature of the traffic and allows the model to again be a good approximation.

To see if the transient behavior of the queue can also be predicted for a different queue utilization, the utilization was reduced to 50%, corresponding to a router service rate of 125,000 bits per second. A utilization of 50% was chosen to provide a noticeable change when compared to a utilization of 85% without reducing the loading to a point at which the nonlinear dynamics become impossible to observe. The theoretical behavior calculated by Equation (3.15) is shown in Figure 4.11. The results of the OPNET simulation are
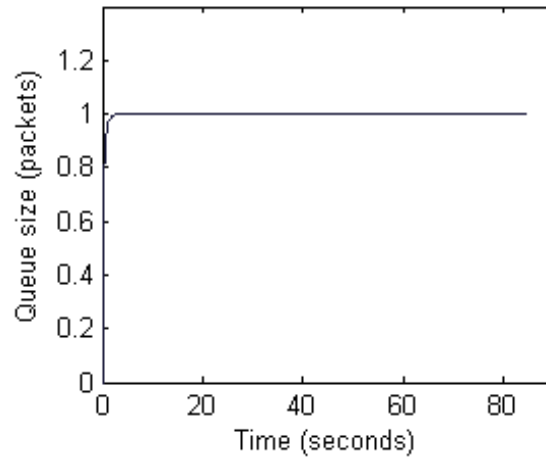
4-8

Figure 4.11   The theoretical expected queue size is shown for an utilization of .50 and a service rate of 10 packets/second
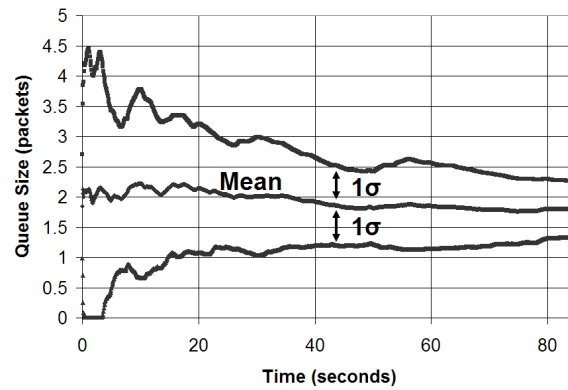

Figure 4.12   Router 1 central queue size in *packets*, 50% loaded, eleven-run Monte Carlo analysis.

shown in Figure 4.12.

Notice that the transient behavior matches the theoretical result including the very fast rise time. The small control packets have less of an effect on queue size due to the small size of the queue. The control packets are infrequently trapped by larger data packets.

4.1.3   Mixed Traffic.   It was seen that, as long as the traffic can be adequately approximated as Poisson, the theoretical transient model provides good results. The Jackson theory states that, when a sufficient number of non-Poisson traffic streams are combined, they tend to become Poisson [11]. To test this theory, an OPNET model was generated
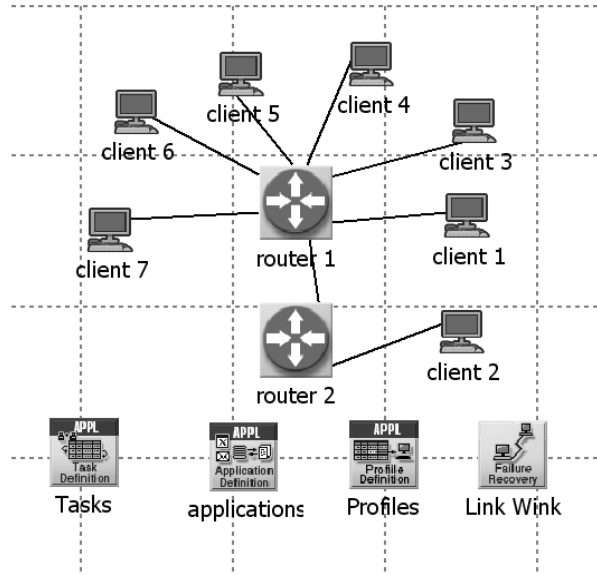
Figure 4.13   OPNET model of the mixed traffic network.

with multiple non-Poisson traffic flows feeding into a router. This network can be seen in Figure 4.13. Again the standard OPNET workstation and router models were used. The standard application models included with OPNET were used to generate traffic. Unlike the earlier scenarios, the traffic is no longer modeled by a single distribution. In this section, each aspect of the application is modeled. For example, in the FTP application model, the PUT and GET messages are modeled along with the data traffic, all of which are sent using a TCP model. Therefore, at any moment in time, the traffic distributions are dependent on many variables including the performance of the transport and routing protocols in the network. Table 4.1 shows the application profiles used by each workstation. Seven workstations were modeled to test if the model was valid for a relatively small number of clients. A reasonable cross-section of various traffic types was chosen to represent a diverse mix. Client 2, which is located in the lower right of the network in Figure 4.13, acted as a server to the other clients. TCP Reno was used by all of the workstations. Again, the RIP routing algorithm was used by the routers.

To observe the results of mixing multiple traffic streams, the queue behavior of router 1 is observed. The raw traffic received by router 1 is shown in Figures 4.14 and 4.15. The average number of bits/second received by all links is 60,985 and the average number of

Table 4.1   Traffic Type Generated By Each Client

Client	Traffic Type
1	Heavy email, heavy database, and heavy FTP
2	Server to other clients
3	Image browsing
4	Heavy browsing, light email, and light database
5	Light browsing and medium database
6	Heavy FTP and heavy email
7	Medium FTP and heavy browsing

packets received is 10.56 packets/second. The service rate of router 1 was set to 71747 bits/second to correspond to a utilization of .85. The individual traffic streams shown in the data plots are non-Poisson.

The theoretical transient response calculated by Eq. (3.15) is shown in Figure 4.16. The result obtained by the OPNET simulation in Figure 4.17 is very close to the theoretical prediction. Notice that the standard deviation is greater in this section than in the previous more ideal scenarios. This is because, unlike in the previous sections, the incoming traffic is only approximately Poisson. Since the mean value obtained from the Monte Carlo analysis is very close to the theoretical, the transient queue model is still valid for this scenario; however, to account for the greater uncertainty, the value of the dynamics noise covariance $\mathbf{Q_d}$ in Eq. (3.17) will have to be increased accordingly. This validates that the transient queue model developed is adequate when the incoming traffic to a queue is a combination of multiple sources.

4.1.4   Model Validation Summary.   In the above sections, the transient queue model was validated for three scenarios of increasing complexity. The mean results of the Monte Carlo analyses performed matched the theoretical results. However, due to the random nature of the Poisson arrival process, the standard deviations given by the Monte Carlo analyses can be significant. Thankfully, the Kalman filter takes into account the unknown dynamics through a proper choice of $\mathbf{Q_d}$. The transient queue model provides the Kalman filter with the very import knowledge of the general trend describing the size of the queue. The fact that these unknown dynamics occur, *underscore* why a Kalman filter is needed to provide adequate estimates and predictions of the state of the system.
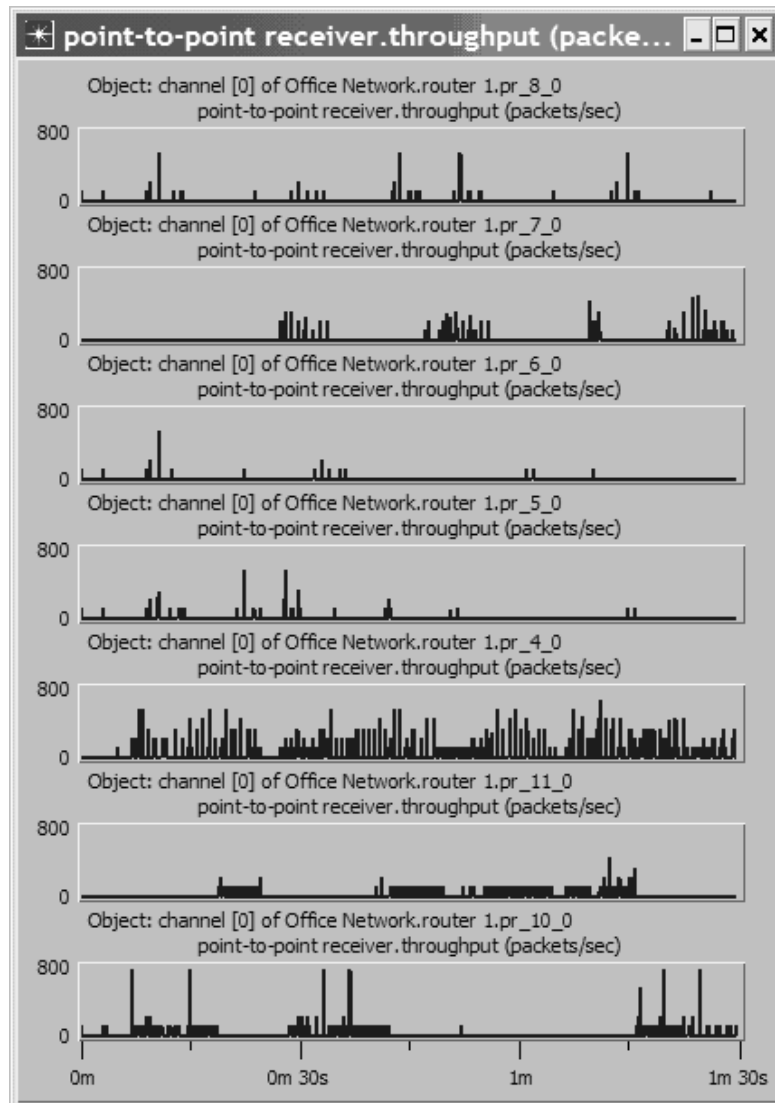
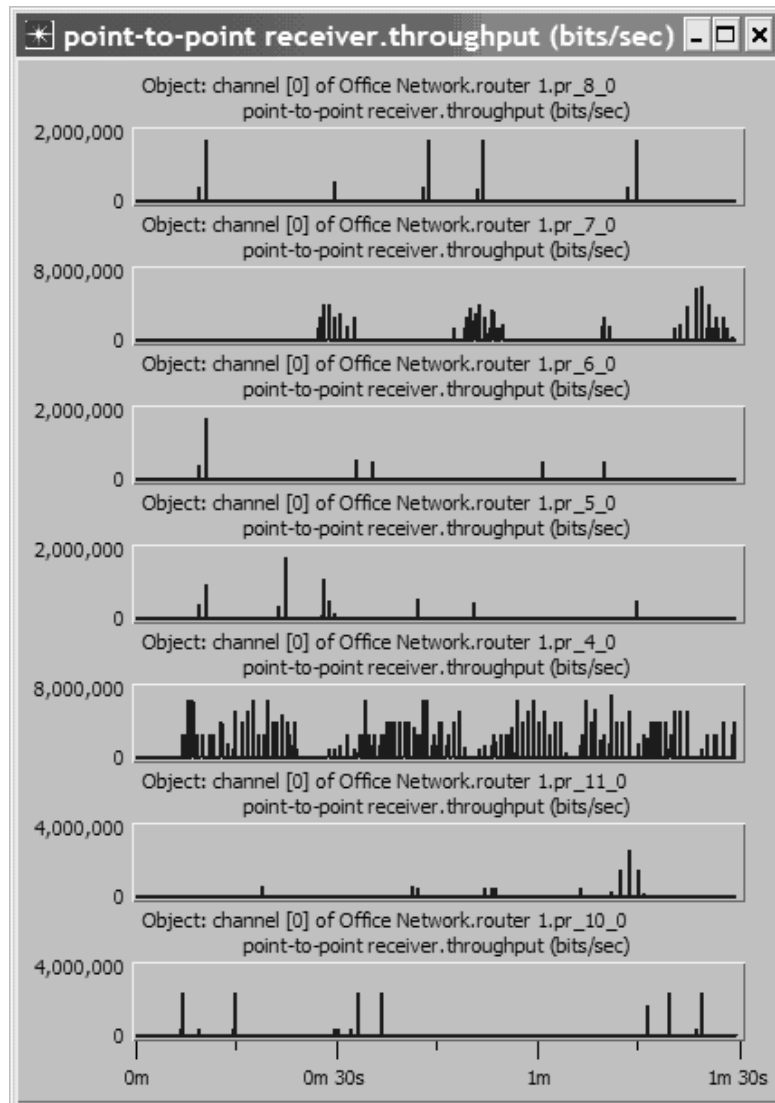Figure 4.14   Traffic received in *packets*/second by router 1 from each incoming link.

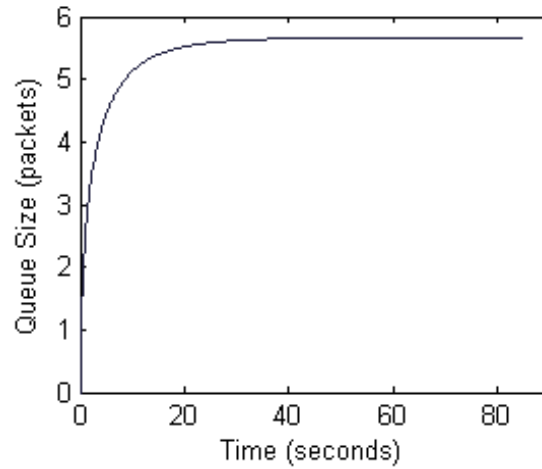Figure 4.15   Traffic received in *bits*/second by router 1 from each incoming link.

Figure 4.16 The theoretical expected queue size is shown for an utilization of .85 and a service rate of 12.4 *packets*/second
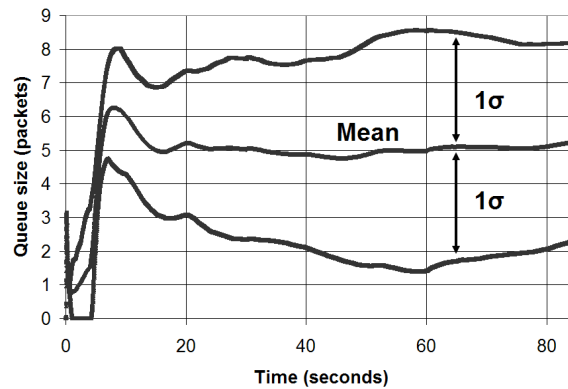

Figure 4.17 Router 1 central queue size in *packets*, 85% loaded, eleven-run Monte Carlo analysis.

Long-range dependant (LRD) or self-similar traffic has commanded a lot of interest in the last decade [23]. It has been observed in varying degrees in many different scenarios within computer networks. LRD traffic is of concern because it does not exhibit long-term smoothness. This leads to a traffic source with increased burstiness over long time scales when compared to Poisson traffic. Using a Poisson dynamics model to characterize LRD traffic over long time scales can result in an under-estimation of queue sizes. However, when LRD traffic is viewed at a sufficiently small scale it appears Poisson [12]. Therefore, the dynamics model associated with Poisson arrivals proposed in this document is valid over small time scales, allowing the model to be used as the basis of the Kalman filter with the understanding that the accuracy of the model decreases as the time scale increases. The uncertainty introduced over time by the LRD behavior is represented by the Brownian motion process describing the uncertainty associated with the dynamics model and is characterized by $\mathbf{Q_d}$ in the Kalman filter equations.

4.2   Network Estimator Performance

4.2.1   *Extended Kalman Filter Validation.*   The Extended Kalman Filter Eqs. (3.23)-(3.27) were implemented in MATLAB version R2006a in order to validate the design of the network estimator. The primary goal was to generate an accurate estimate of queue size given noise-corrupted measurements. The measurement data was generated by the mixed traffic OPNET simulation described above. The number of packets contained in the central queue of router 1 was sampled and the data file was manually transferred to MATLAB. Within MATLAB, a noise with a variance of R=10 was added to the sampled data to simulate the measurement uncertainty that would be found in a real world network. The primary source of measurement error within computer networks is due to the lack of a global time reference. Since the time at which a measurement was taken in not known exactly, the amount of measurement noise is directly related to the relative accuracy of the clocks within network components. An R=10 was estimated to provide a rough approximation of this measurement noise.

Initially, the sample rate was set to 10 seconds. To obtain an adequate estimate, the filter must be tuned properly. This is done by adjusting $\mathbf{Q_d}$ in an iterative manner
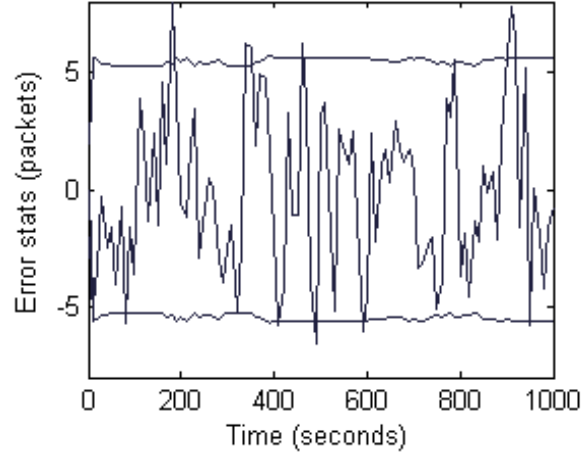
Figure 4.18   Filter error statics corresponding to the queue size state variable x_1

until the desired performance is achieved. To aid in this process, it is useful to plot the filter error and the filter-computed two-sigma values of the error on the same plot for each state variable. The filter error is computed by subtracting the filter estimate from the actual value obtained from the OPNET simulation. The two-sigma values of the error are obtained by taking the square root of the variance term corresponding to the state variable of interest obtained from the filter computed covariance matrix, and then multiplying by two. The diagonal values of the $\mathbf{Q_d}$ matrix are then adjusted until 95% of the filter error values fall within the two-sigma bound. An example for a well-tuned filter is shown in Figure 4.18. For a sample period of 10 seconds, a $\mathbf{Q_d} = \begin{bmatrix} 20 & 0 \\ 0 & .01 \end{bmatrix}$ was found to give adequate performance. The estimate of queue size in packets is given in Figure 4.19. The estimate is overlaid on the actual queue size value. The network estimator results shown here are only single sample runs. It would be desirable to perform a Monte Carlo analysis of the network estimator performance, however this was prohibitively difficult in the time given due to limitations in the network simulation tool. The simulation used a global random seed value and any change in the simulation would result in totally different queue sizes. Numerous performance runs have been completed and the results given are indicative of the normal operation of the network estimator.
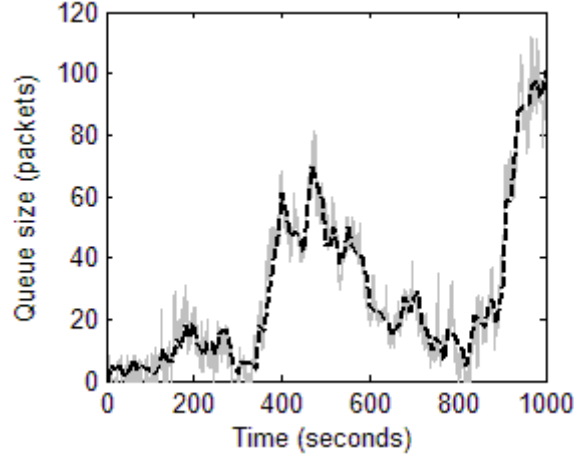
Figure 4.19   Estimate of queue size in *packets* overlaid on the actual queue size values, 10 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate

Figure 4.20   Average packet size in the queue

Notice that the estimate of queue size provided by the network estimator does a very good job of following the actual queue size. It is impossible to predict the high frequency behavior of the queue, so one can only hope to achieve an accurate track of the low frequency behavior.

It is also desirable to estimate the queue size of the router in bytes. For this to be achieved, the estimate of queue size in packets can be multiplied by the average packet size contained in the queue. The average packet size in the queue during a simulation run is shown in Figure 4.20. The average packet size changes rapidly and varies significantly

4-17
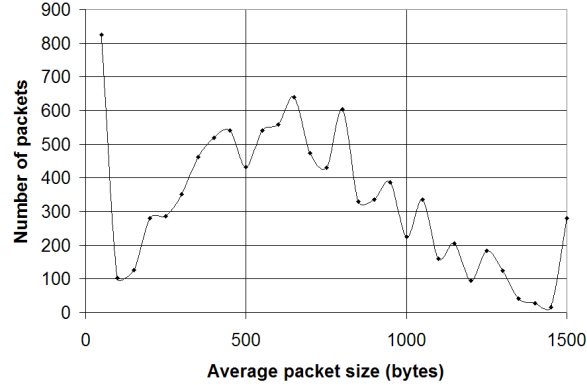
Figure 4.21   Histogram of average packet size in the queue

between 40 and 1500 bytes. To gain a better understanding of the average packet size in the queue, a histogram of the data is shown is Figure 4.21. With the exception of the tails on either end, the distribution is almost Gaussian. This distribution can be explained by the fact that almost all packets in the network are either full size data packets or small network control packets. Sometimes the network queue contains only the small control packets or the large data packets, which represent the tails of the distribution. The Gaussian-like part of the distribution represents the different mixtures of large and small packets that occur. The average packet size in the queue over all time was found to be 550 bytes. The estimate of queue size in packets was multiplied by 550 bytes and compared to the actual queue size in bytes in Figure 4.22. Using only a single value for the average packet size to estimate the queue size in bytes worked very well. The estimate of the queue size in bytes does a very good job of tracking the actual values, except near the end of the simulation when the average packet size decreased below 550 bytes.

To see the effect of reducing the sample rate, the simulation was repeated with a sample period of 100 seconds. The iterative filter tuning procedure described earlier was conducted and $\mathbf{Q_d} = \begin{bmatrix} 30 & 0 \\ 0 & .1 \end{bmatrix}$ was found to give the best performance. The estimate of the number of packets contained in the queue can be seen in Figure 4.23. Even at a much reduced sample rate, the filter estimates the queue size adequately. The queue size estimate in bytes is shown in Figure 4.24 and also continues to be an adequate estimate at the reduced sample rate.

Figure 4.22   Estimate of queue size in *bytes* overlaid on the actual queue size values, 10 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate
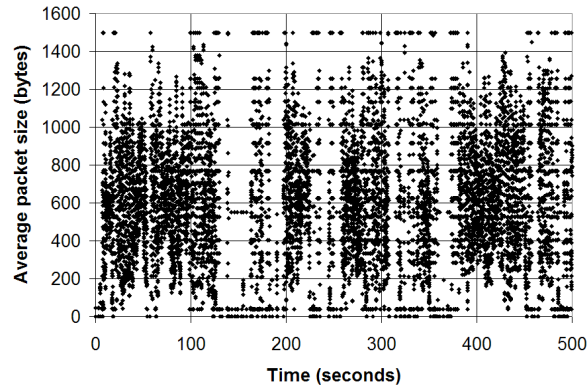

Figure 4.23   Estimate of queue size in *packets* overlaid on the actual queue size values, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate
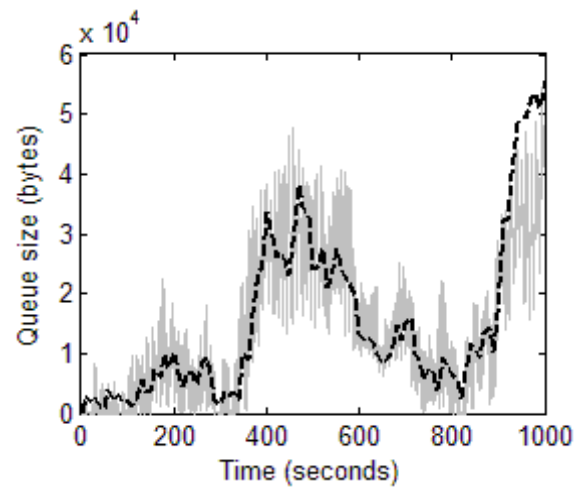
Figure 4.24   Estimate of queue size in *bytes* overlaid on the actual queue size values, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate
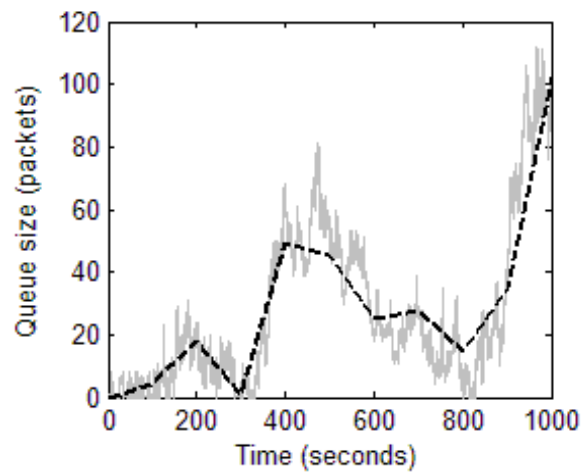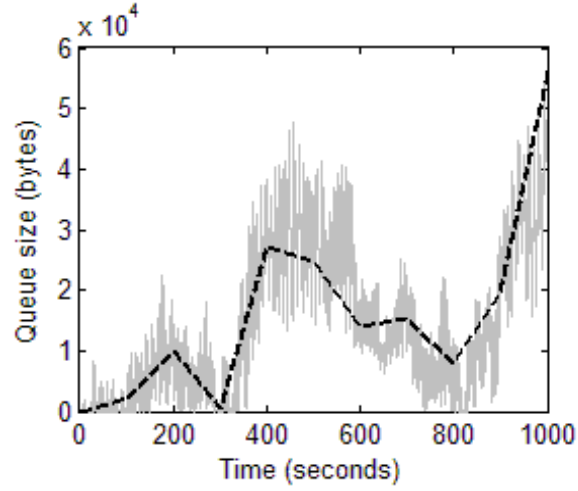
Another advantage of the Kalman filter is its predictive ability. The $\hat{x}_1(t_i^-)$ values, which represent the time-propagated estimate of queue size before the measurement update, are plotted in Figure 4.25. The prediction lags in a number of places due to the very low sample rate. It is impossible of predict a sudden change in queue size without measurements to indicate the changing behavior. While not perfect, the filter is a better predictor of queue size behavior 100 seconds into the future as compared to other methods. For instance, given measurements at 300 and 400 seconds, a linear interpolation would have predicted a much less accurate value at 500 seconds than the Kalman filter generates. The prediction is also more accurate than performing a zero order hold on measurements between sample times.

4.2.2   Integrated Network Estimator.   The Kalman filter validated above was integrated into the OPNET simulations to demonstrate the operation of a network queue estimator in a computer network. Since the Kalman filter was implemented in MATLAB and the network simulation was implemented in OPNET, a co-simulation interface was developed between the two programs. OPNET was used as the primary program and maintained control of the simulations. When Kalman filter computations were needed, the MATLAB functions were called by the OPNET simulation. A C application programing
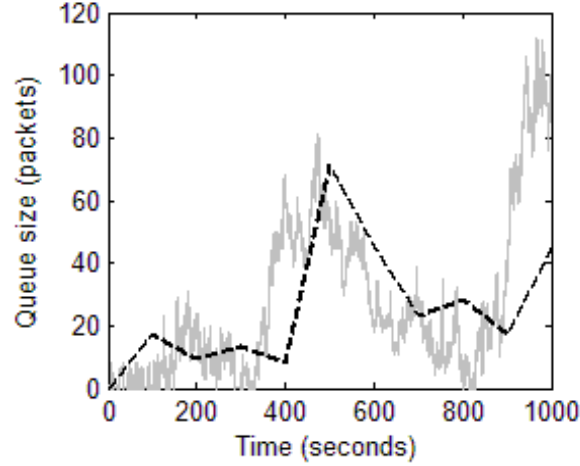
Figure 4.25   Prediction of queue size in *packets* overlaid on the actual queue size values, 100 second sample period, grey line: actual queue size, overlaid black dashed line: Kalman filter estimate

interface (API) that allows the MATLAB engine to be called by C code is included with the MATLAB software. Since OPNET models are constructed of C code, this API was used to call MATLAB from within the OPNET models. For specifics on how to interface the two programs, refer to Appendix A.

The integrated network estimator model is shown in Figure 4.26. The router and workstations models are modified versions of the models used in Sections 4.1.2 and 4.1.3. The internal model of the router is shown in Figure 4.27. The IP block has been modified to observe the IP packets traveling through the router. Each IP packet is checked to see if it contains a TCP packet. If the IP packet contains a TCP packet, the source and destination IP address and the source and destination port numbers are recorded. These four pieces of information uniquely identify a TCP stream. This packet identification information is sent to the queue size monitor block. The size of the central queue in the router is also sent to the queue size monitor block from the IP block.

The queue size monitor block has been added to the router model. The queue size monitor block contains the queue size monitor application. The function of the queue size monitor application is to send the router queue size data to all of the senders that are transmitting data through the queue. Each of these senders has a unique IP address. The queue size monitor consists of a root process and a child process. The root process creates
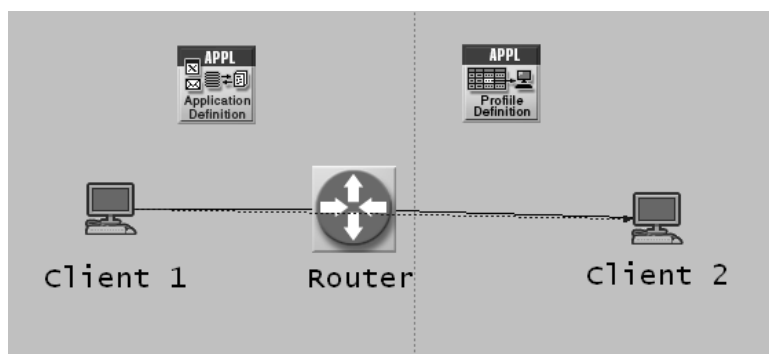
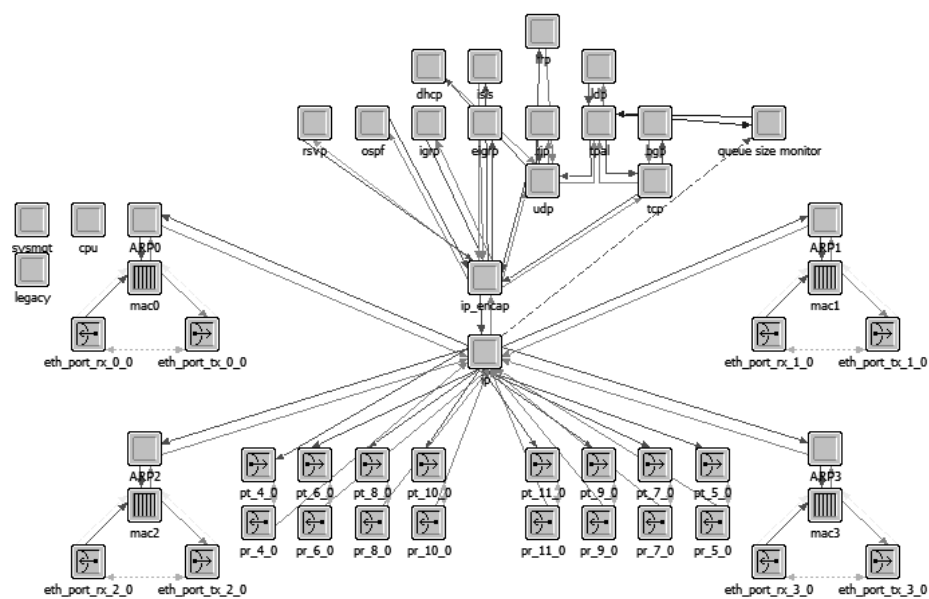Figure 4.26   Integrated network estimator model

Figure 4.27   Modified router model

a child process for each unique IP address. The child process is responsible for sending the queue size data to the IP address associated with it. When the root process receives the packet identification information, it checks the source IP address of the packet. If a child process already exists for this IP address, the packet identification information is passed to the child process. If a child process does not exist for this IP address, one is created and the packet identification information is sent to the newly created child process.

When a child process is created, it registers itself with the Transport Protocol Adaptation Layer (TPAL). TPAL is the interface OPNET uses to communicate between the application layer and the transport layer. A custom queue report packet is created by the child process and contains two data fields: queue size and sample time. The queue size field contains the size of the queue in packets and the sample time field contains the time at which the queue was sampled. The rate at which the child process sends these custom queue report packets is governed by the queue size sample rate parameter and can be varied by the user. Each individual child process is responsible for sending queue size data to one sender. The child process keeps track of the sender's individual TCP connections travelling through the router. The individual TCP connections are identified by the port number. The individual TCP connections are tracked by creating a list containing the port numbers corresponding to active TCP connections. The time of the last packet received by the router for an individual port number is recorded as part of this list. When a child process receives the packet identification information from the root process, it checks to see if the source port number is in the list. If it is in the list, the timestamp is updated, and if the port number is not in the list, the port number is added along with the timestamp. A port number is removed from the list if a packet from the corresponding TCP stream has not been observed by a timeout period which is a parameter set by the user.

Before a packet is sent by a child process as governed by the queue size sample rate, the port list is checked to see if any of the timestamps are older than the timeout period. If this is the case, the ports are removed from the list. If after removing the applicable ports, the list is empty, then the child process is destroyed. However, if the list still contains ports denoting active connections, then the queue size data packet is created and passed to
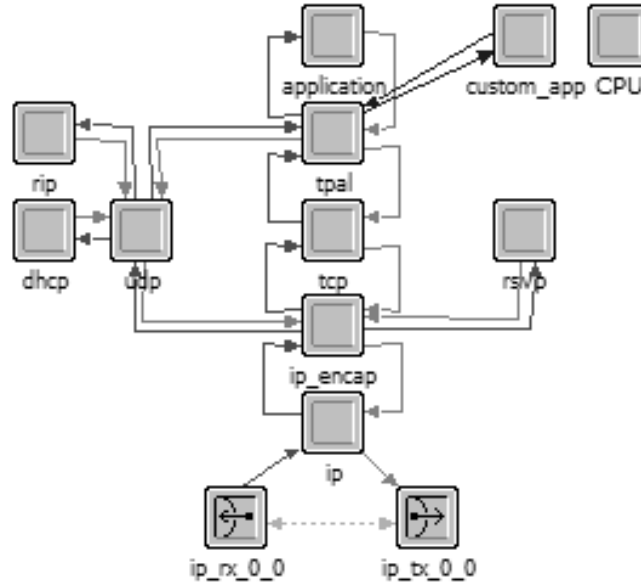
Figure 4.28   Modified workstation model

the TPAL interface, which sends the packet to the IP address corresponding to the child
process.

The workstations receive queue size data packets from the routers through which
their TCP connections pass. The modified workstation model is shown in Figure 4.28.
The Kalman filter block has been added. When the workstation receives a queue size
data packet, the TPAL interface forwards it to the Kalman filter block. The Kalman filter
block receives the queue size data packet and calls the Kalman filter MATLAB code that
was validated in Section 4.2.1 through the co-simulation interface. The Kalman filter code
processes the measurement data by executing the propagate and update steps given in
Section 3.3.1. The estimate of queue size and packet arrival rate is then passed back to
OPNET and recorded.

To test the integrated network estimator, Poisson background traffic with exponen-
tially distributed packet sizes was sent from client 1 to client 2. The mean packet size of
the background traffic was 550 bytes with the arrival rate shown in Table 4.2. An FTP

Table 4.2   Background traffic

Time (seconds)	Arrival rate to queue
0-900	45,000 bits/second
900-1100	30,000 bits/second
1100-1500	32,000 bits/second
1500-2000	44,000 bits/second

transfer was set up between clients 1 and 2 using the standard FTP application model included in OPNET. Two 500 kilobyte files are sent: one at 50 seconds and one at 1050 seconds. The queue size sample period was set at 1 second and the timeout period was set to 2 minutes. Figure 4.29 shows the Kalman filter estimate made by client 1 when $\mathbf{Q_d}$ $= \begin{bmatrix} 15 & 0 \\ 0 & 1 \end{bmatrix}$ and R = 10. The actual queue size is shown in grey and the estimate of queue size is overlaid in black. Notice that an estimate of the queue size is made by client 1 only when client 1 is sending data through the queue. The file transfers are completed at 150 and 1335 seconds, but the estimate of queue size is continued for another 120 seconds until the timeout period expires. The estimate of queue size made by the Kalman filter tracks the actual queue size very closely.

In some cases, it is desirable to track the low frequency behavior of the queue. This can be accomplished by increasing the R value, which places less weight on the measurements and more weight on the transient queue model which does not include the random high frequency behavior. To demonstrate the ability to track only the low frequency behavior of the queue, the above experiment was repeated with an R value of 1000 and is shown in Figure 4.30. The network estimator was successful in tracking only the low frequency behavior of the queue.

To demonstrate the ability to track low frequency behavior further, the above experiment was repeated without background traffic. The actual queue size is shown in Figure 4.31. To clarify the graph, the line is not drawn when the queue size changes. The actual queue size is fluctuating between 2 and 8 packets rapidly. When R =1, the Kalman filter tracks the actual queue size accurately, as seen in Figure 4.32. Again, to clarify the graph, only the discrete estimates are plotted. When R is increased to 250, only the low frequency behavior of the queue is tracked as observed in Figure 4.33.

Figure 4.29   Estimate of queue size by Client 1 overlaid on the actual queue size values,
                background traffic enabled, R = 1, grey line: actual queue size, overlaid
                black line: Kalman filter estimate
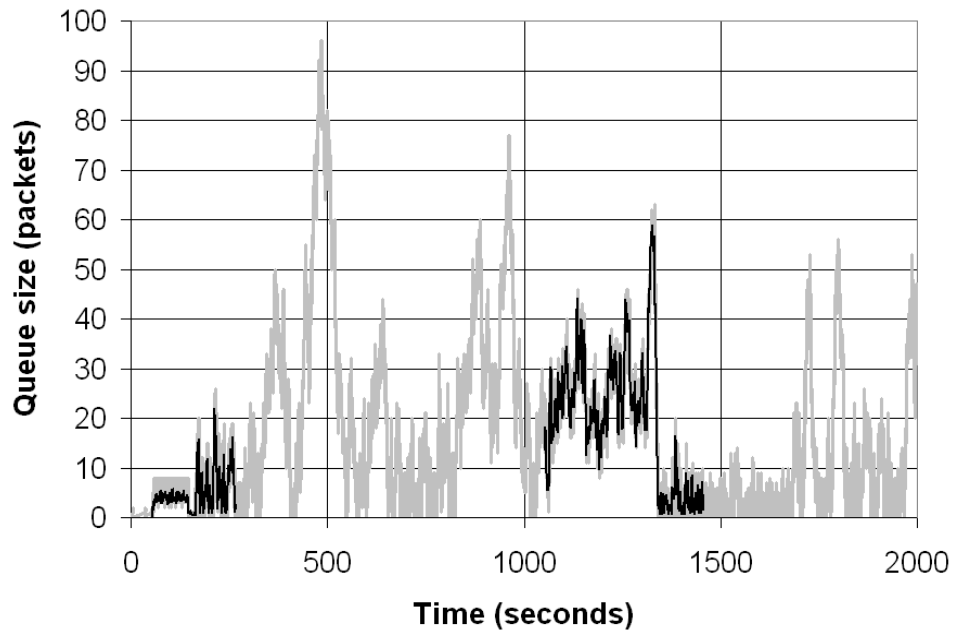

Figure 4.30   Estimate of queue size by Client 1 overlaid on the actual queue size values,
                background traffic enabled, R = 1000, grey line: actual queue size, overlaid
                black line: Kalman filter estimate
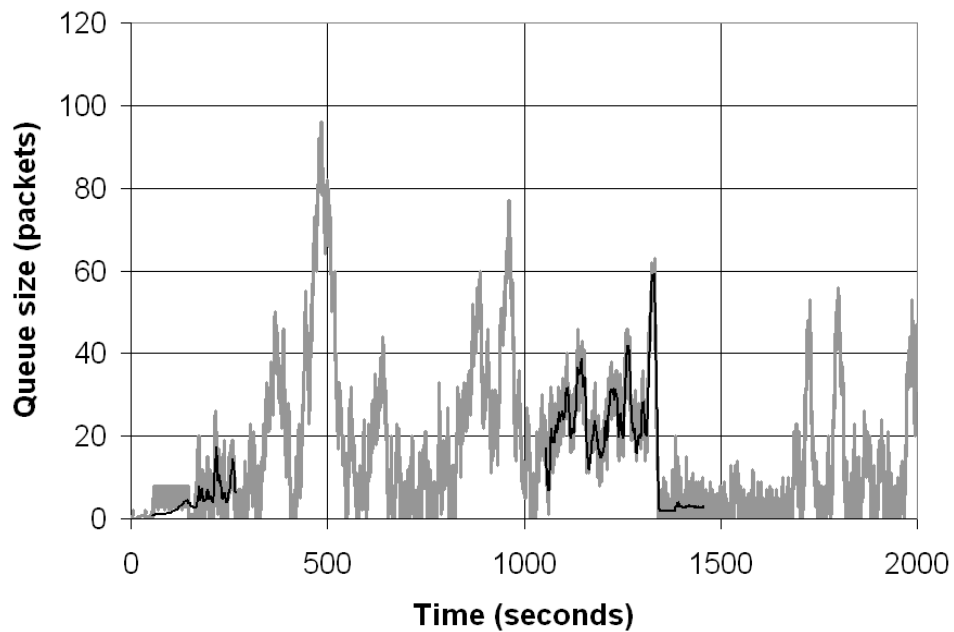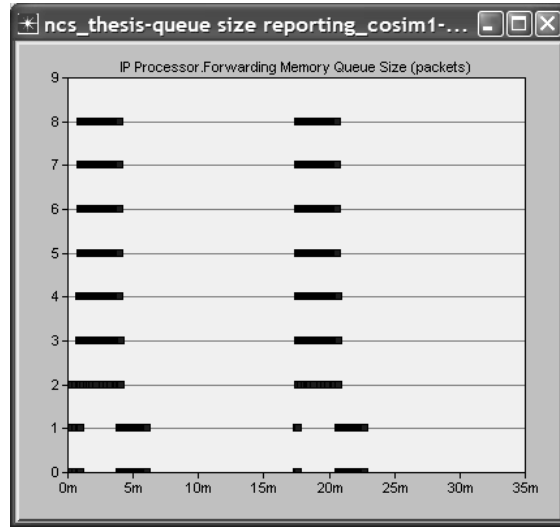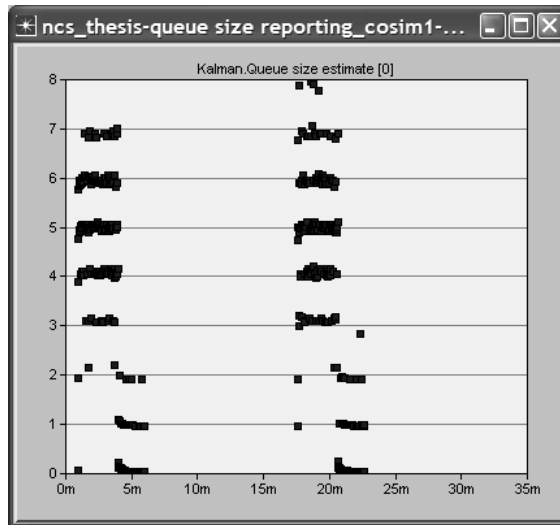
Figure 4.31   Actual queue size, no background traffic

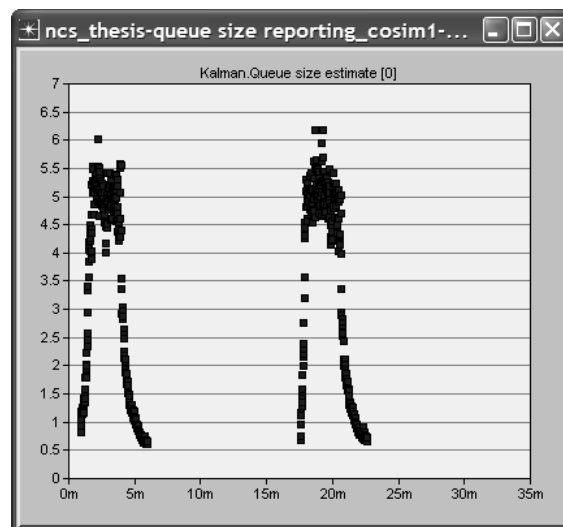Figure 4.32   Estimate of queue size by Client 1, no background traffic, R = 1

Figure 4.33   Estimate of queue size by Client 1, no background traffic, R = 1

4.3 Simple Network Queue Controller

The deterministic LQG *steady state* linear perturbation PI controller designed in Section 3.4.2 is implemented in MATLAB. The simple OPNET model used to test the controller design is shown in Figure 4.34. The two dashed lines are statistic lines that report the queue size and queue arrival rate to the source. The source and queue models are modified versions of the standard OPNET models used in Section 4.1.1 and the sink model remains unchanged. The queue model has been slightly modified to record the arrival rate to the queue. The source model has been modified so that the sending rate is no longer set by the user, but computed by the controller. Since the controller is implemented in MATLAB, the source interacts with the controller through the co-simulation interface. The constant gains $\bar{\mathbf{G}}_{\mathbf{c}}^* = \begin{bmatrix} \bar{\mathbf{G}}_{\mathbf{c1}}^* & \bar{\mathbf{G}}_{\mathbf{c2}}^* \end{bmatrix}$ and E are computed at the beginning of the simulation given the setpoint y_d and tuning parameters, X_{11}, X_{22}, X_{33}, and U. The Riccati equation given by Eq. (3.41) is solved using the discrete-time algebraic Riccati equation (DARE) solver in MATLAB. Once the constant gains are calculated, the simple control law given by Eq. (3.56) is executed in MATLAB. The controller calculates the packet send rate, based on the queue size and queue arrival rate information. In this experiment, the Kalman filter is not used since the controller is given direct access to the state variables. The source has an unlimited number of packets available to send. The packet sizes are exponentially distributed with a mean size of 1500 bits.

Multiple simulations were conducted in which order of magnitude changes were made to the tuning parameters to understand the effect on the performance of the controller. X_{11}, X_{22}, X_{33}, and U were varied by orders of magnitude between 1 and 10000 resulting in 625 simulation runs. As stated in Section 3.4.2, the weights X_{11} and X_{22} correspond to the importance of maintaining the states at the nominal values, the weight X_{33} corresponds to the importance of minimizing the regulation error, and the weight U corresponds to the importance of minimizing control action. The simulations were performed for 100 seconds of simulated time with a queue service rate of 10000 bits/second. The setpoint was set to the relatively low value of 3.125 packets. The setpoint was set purposely low as a challenge to the controller, because small queue sizes are more difficult to maintain accurately. A small queue size is hard to maintain because the smallest deviation of one packet is a high
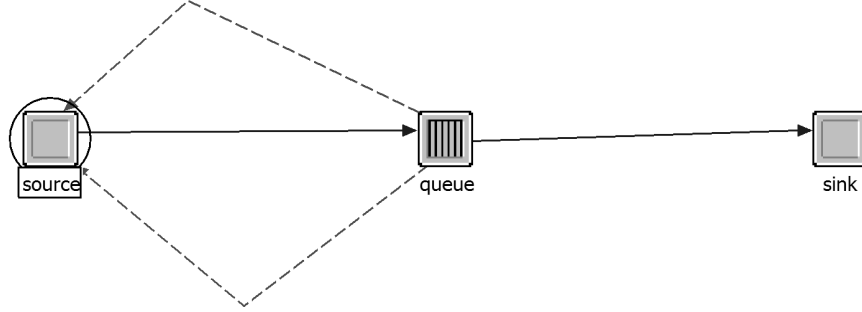
Figure 4.34   Simple controller model

percentage of the overall queue size. The ratio of U to X_{33} was found to be the most important factor. It was observed that, in order to provide accurate control of the queue, U must be multiple orders of magnitude larger than X_{33}. The U to X_{33} ratio of 1000 was found to provide adequate results. The controller was found to be fairly robust to changes in X_{11} and X_{22}.

Figures 4.35 - 4.38 display the results of the simple network queue controller under different circumstances. In all of these graphs, a 30-run Monte Carlo analysis was conducted. In these figures, Graph (a) displays the queue arrival rate in packets/second, which also corresponds to the output of the controller, Graph (b) displays the queue size in packets, and Graph (c) displays the queue size in bits. The middle line is the mean and the outside lines correspond to the mean plus or minus one standard deviation of the data displayed.

Weights of $\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and U = 1000 resulting in $\bar{\mathbf{G}}_{\mathbf{c}}^{*} = \begin{bmatrix} .1064 & .3842 & .0248 \end{bmatrix}$

were found to provide good results, as shown in Figure 4.35. The controller is able to maintain the setpoint of 3.125 with a standard deviation of around 2.5 packets. The devi-

ations from the setpoint are due to the variability of the packet sizes, which the controller is unable to respond perfectly.

In the second case, X_{22} was increased from 1 to 100, resulting in controller gain values of $\bar{\mathbf{G}}_{\mathbf{c}}^* = \begin{bmatrix} .1041 & .4415 & .0236 \end{bmatrix}$. The controller's robustness to change in this parameter are seen in Figure 4.36. The results are very similar to the previous case. The steady state performance remains virtually unchanged, with the only noticeable difference being that the rise time has been slightly increased.

The effect of decreasing the U to X_{33} ratio is shown in Figure 4.37. The original values of $\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ are used, but U is decreased to 1, resulting in gain values of $\bar{\mathbf{G}}_{\mathbf{c}}^* = \begin{bmatrix} .8880 & .8936 & .3262 \end{bmatrix}$. Since the cost associated with control inputs is decreased, the controller tries to respond to high frequency variations in the queue size in a more aggressive manner. This results in overcontrol and the deviations from the setpoint actually increase. Since the negative variations are clipped by an empty queue size, the large positive variations result in a average queue size that is approximately two packets larger than the setpoint. Also, the positive standard deviation is increased to around 7 packets.

To show the effect of changing the setpoint, the original weights $\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and U = 1000 are used and y_d is set to 53 packets, resulting in $\bar{\mathbf{G}}_{\mathbf{c}}^* = \begin{bmatrix} .1610 & .4728 & .0230 \end{bmatrix}$. The result of increasing the setpoint is shown in Figure 4.38. The variance is actually slightly greater than in Figure 4.35; however, the variance is a much smaller percentage of the setpoint value. The sending rate computed by the controller in Graph (a) spikes as the queue is filled and then levels off, resulting in a smaller rise time. The controller correctly responds to the nonlinear dynamics of the queue as highlighted by this experiment.

Figure 4.35   Simple network queue controller results, $y_d = 3.125$, $X_{11} = 1$, $X_{22} = 1$, X_{33}
$= 1$, and U = 1000, 30-run Monte Carlo analysis, black line: mean, gray
lines: mean plus or minus one standard deviation
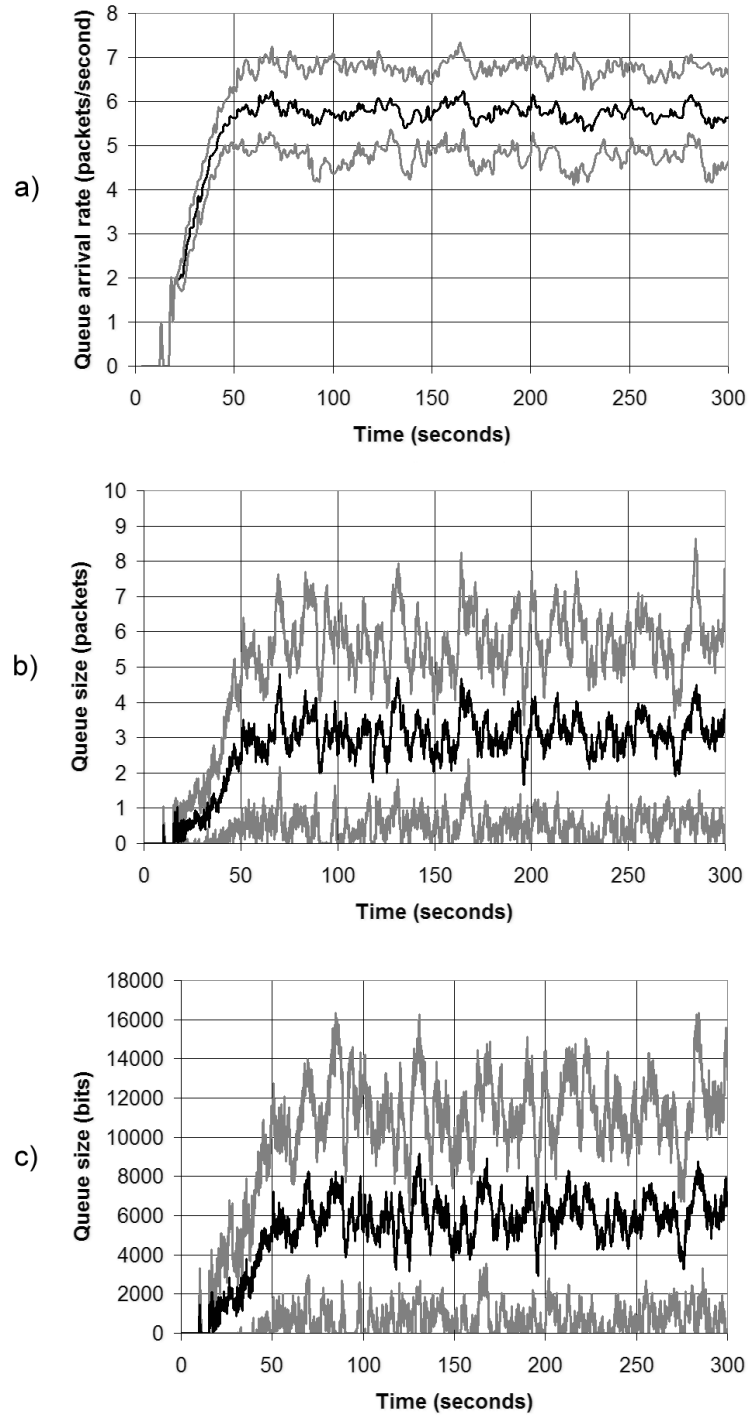
Figure 4.36 Simple network queue controller results, $y_d = 3.125$, $X_{11} = 1$, $X_{22} = 100$, $X_{33} = 1$, and $U = 1000$, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation

4-33

Figure 4.37   Simple network queue controller results, $y_d = 3.125$, $X_{11} = 1$, $X_{22} = 1$, $X_{33} = 1$, and $U = 1$, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation
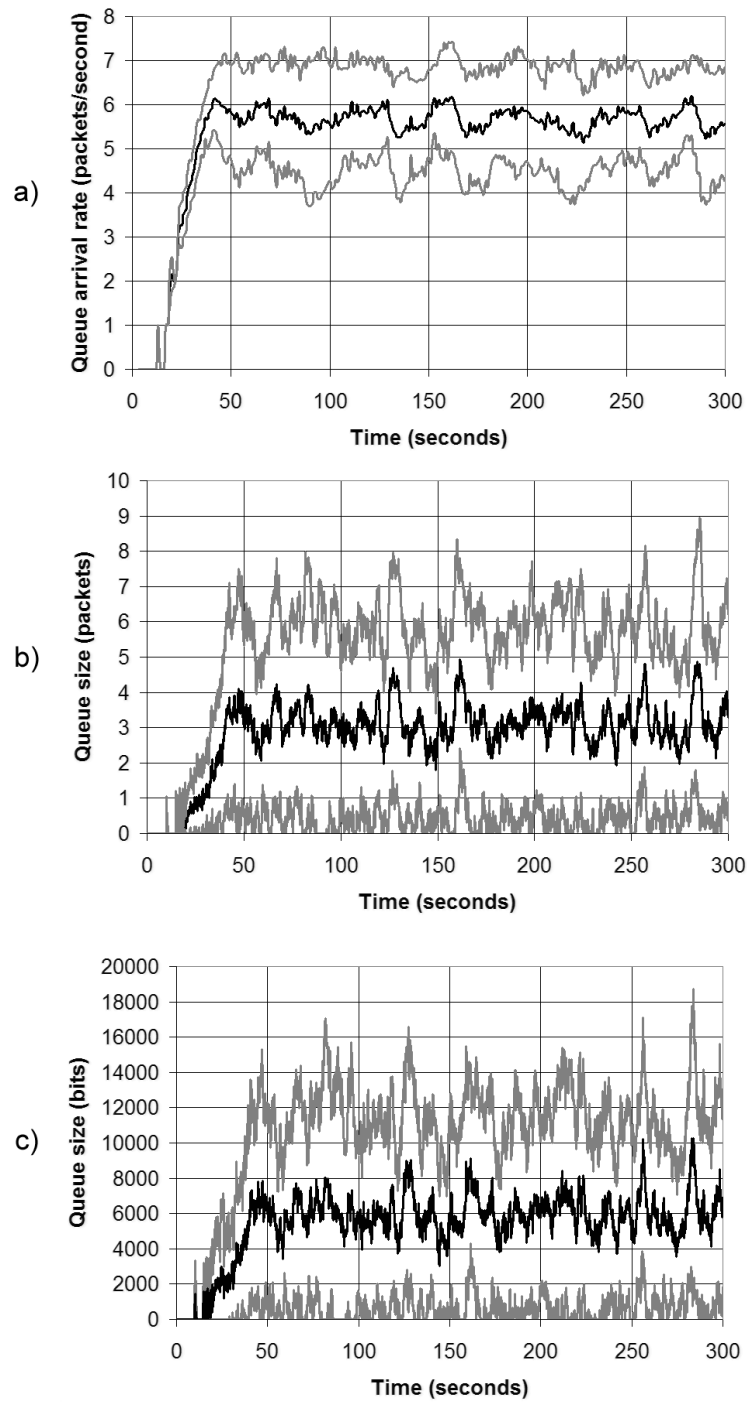
Figure 4.38   Simple network queue controller results, $y_d = 53$, $X_{11} = 1$, $X_{22} = 1$, $X_{33} = 1$, and $U = 1000$, 30-run Monte Carlo analysis, black line: mean, gray lines: mean plus or minus one standard deviation
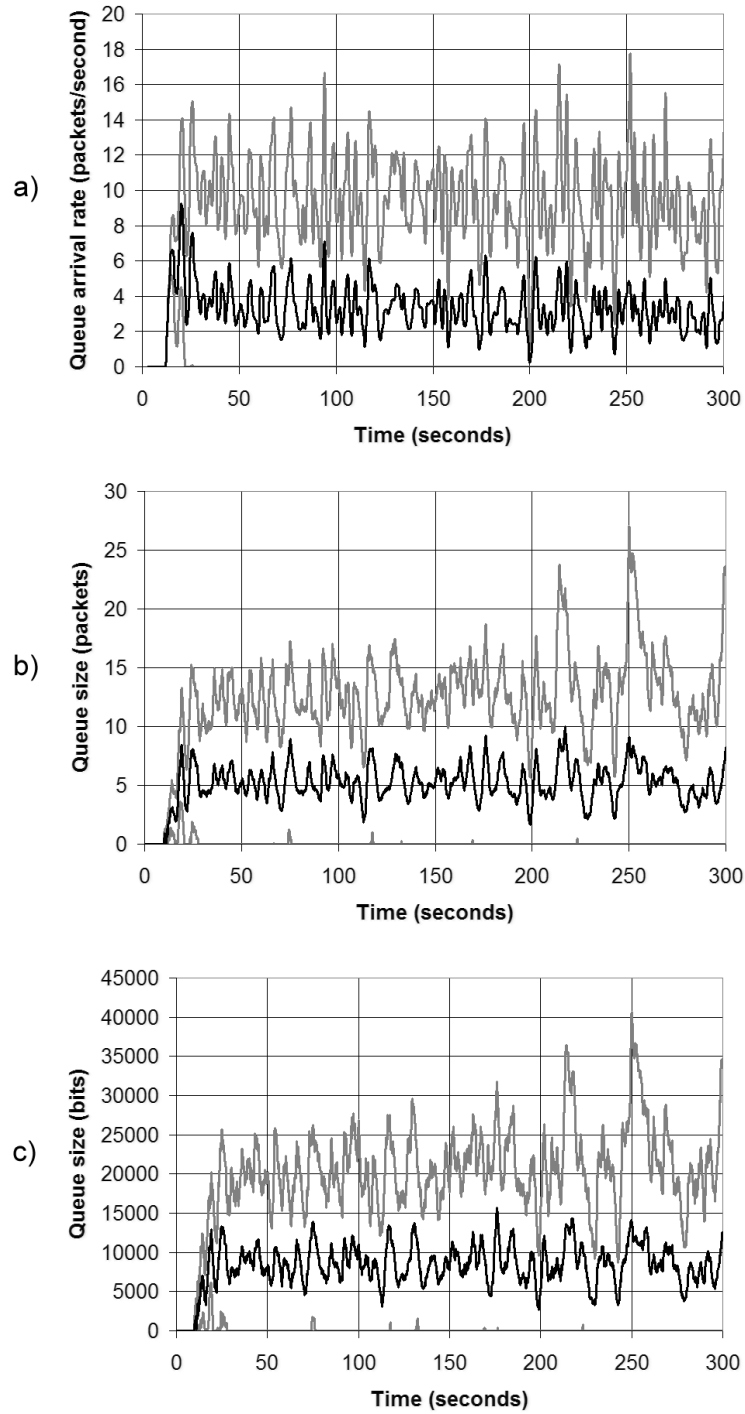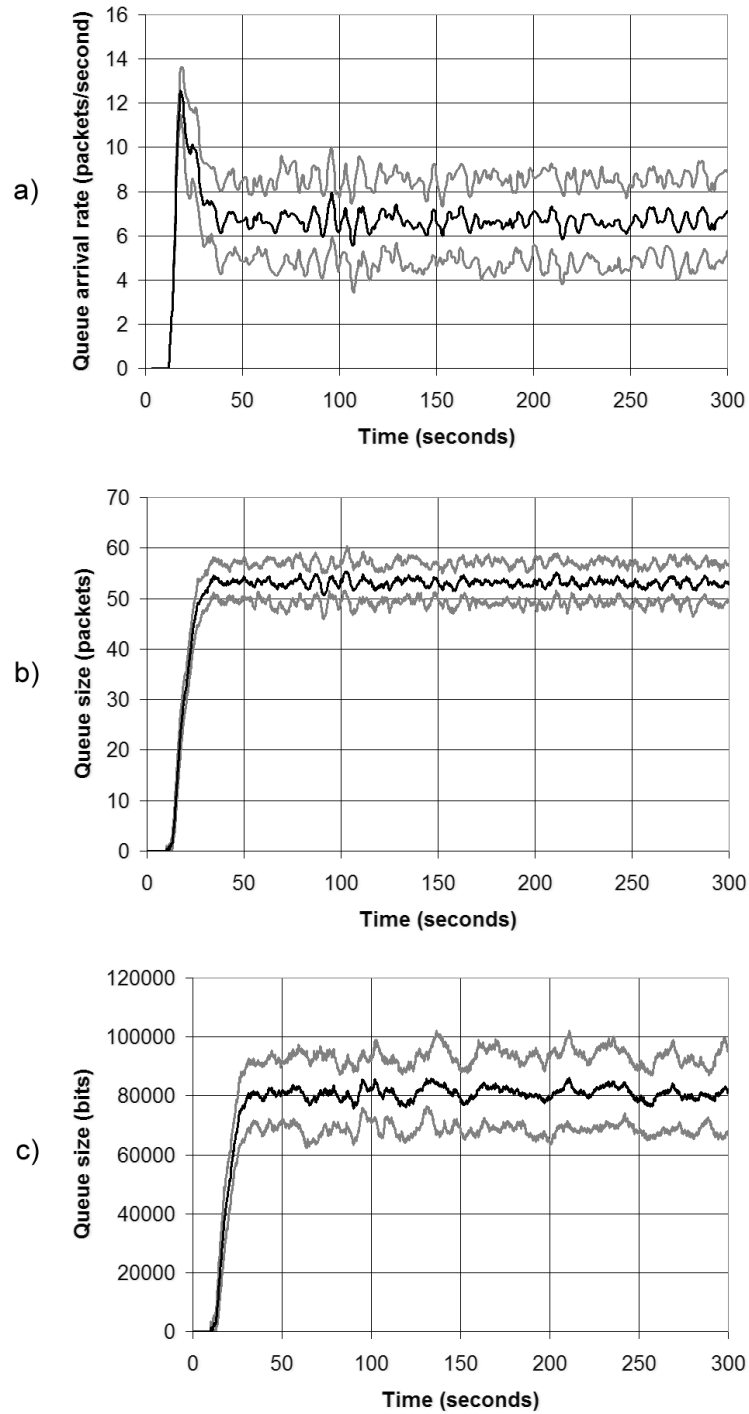
4.4 Summary

In this chapter, the transient queue model was verified both in theoretical and realistic environments. The network state estimator was shown to accurately estimate queue size given noise corrupted measurements. It was also shown that, by changing the tuning of the estimator, the low frequency behavior can also be estimated. The performance of the network queue controller was investigated. The behaviors of a properly and improperly tuned filters were shown. The most important aspect of tuning the controller was found to be the U to X_{33} ratio. Due to the large degree of randomness exhibited by the queue size, this ratio must be sufficiently large to provide dampening and prevent over-control. A properly tuned network queue controller was found to adequately regulate queue size to the given setpoint.

V. Conclusions

5.1 Contributions

In this thesis, stochastic estimation and control theory was extended to computer networks successfully. Due to the largely undeveloped nature of this field, much of this document was devoted to developing the concepts and basic designs of the needed components. Sufficient experiments were conducted to verify the validity of the designs.

A state space network model was adapted from the existing queueing theory equations presented in Section 3.2. Faced with the task of modeling a computer network, the decision was made to focus on queues, since they are the defining component of computer networks. Existing developments in queueing theory were researched and the transient equations describing a queue with an exponentially distributed service rate receiving Poisson traffic were used as the basis for the state space model [3, 6, 11, 20].

In Section 4.1, the transient queue model was validated for multiple scenarios. The model was first verified under ideal conditions that exactly matched the input traffic assumptions made during the development of the model. The network queueing model was then verified by a more detailed network simulation that involved complex models of network hardware operating under ideal traffic situations. More realistic traffic was simulated to validate the queue model further. The model was found to be adequate in the non-restrictive case in which traffic can be approximated as Poisson or is a combination of multiple sources.

A two-state extended Kalman filter network estimator was developed in Section 3.4.1 using the state space transient queue model. The performance of the network estimator was shown in Section 4.2. The network estimator provided accurate estimates of queue size and packet arrival rate, given noise-corrupted measurements of queue size. The network estimator predicted future behavior of the queue more accurately than other techniques and estimated the low frequency behavior of the queue.

The network estimator was implemented in a realistic environment. A router was modified in a simulated network to provide measurements to the workstations sending data through the queue. The extended Kalman filter was inserted into a modified workstation

model and provided estimates of the router's queue size and packet arrival rate based on the received measurements.

In Section 3.4.2, an LQG steady state linear perturbation PI controller was developed to calculate the sending rate needed to regulate queue sizes. The performance of the controller was demonstrated in Section 4.3. The controller responded to the nonlinear nature of the queue effectively, resulting in an accurate regulation of queue size. The network queue controller provides improved control when compared to the simplistic congestion control algorithms presented in Section 2.4.2 that are used by TCP today. Since the simplistic control algorithms used by TCP have only limited knowledge of the network, the sending rate fluctuates undesirably as the simplistic control schemes struggle to send data quickly without congesting the network, as seen in Figure 2.7. Due to limitations in current transmission control schemes, congestion must be created before it can be detected, resulting in less than optimal results. The network queue controller presented in this document provides a great deal of flexibility and the queue size can be regulated to meet the needs of the network.

5.2   Recommendations

This document laid the foundation to control a computer network using stochastic estimation and control techniques. Numerous extensions to this work are needed in order to produce a feasible real world design. This research has largely been a proof of concept effort and many implementation issues must still be considered.

The network simulation used to test the controller in Section 4.3 was very basic and the controller needs to be tested in a more realistic situation similar to the network used to test the estimator in Section 4.1.3. In the more realistic network, the performance of the network controller should be compared to the performance of the existing transmission control protocols described in Section 2.4.2. In the current controller design, a network with only one queue is considered and the framework needs to be designed to expand the existing design to multi-queue scenarios.

The network estimator needs more rigorous testing in multiple network scenarios. Due to time constraints, investigation of the network estimator's performance was limited to a few network simulations. The effect of multiple network configurations and various traffic conditions should be explored. Scalability investigations should be conducted to study how much queue size measurement packets load the network as its size increases.

A study should be conducted to determine the optimal controller setpoint value in various situations. The optimal setpoint value will vary depending on the goals of the designer. If end-to-end delay is of utmost importance, small queue sizes are optimal, while larger queue sizes would be desired if data throughput is of greatest concern.

Fairness of the network control scheme is a major factor than needs to be investigated. When multiple TCP stream are utilizing one queue, it must be ensured that all streams are given fair access and that one stream does not block others. A scheme must be developed that allows controllers to coordinate among themselves to allocate the fair share of network resources.

The basic concept of estimating computer network parameters stochastically should be explored further. A particular issue was investigated in this document and a solution was developed to demonstrate the benefit of extending stochastic estimation techniques to computer networks. There are numerous areas waiting to be explored in this field, and the basic ideas developed in the document are not unique to the congestion control issue explored here. Kalman filter based estimators could be developed to estimate any network parameter imaginable, to include network delay, bandwidth, utilization, bit error rate, throughput, etc. Similarly, the use of stochastic controllers should not be limited to the queue regulation scheme presented in this document. Feedback controllers based on stochastic estimates could be used to control any aspect of network behavior, to include routing and adaptive network reconfiguration decisions.

Appendix A. Interfacing MATLAB with OPNET

A.1 Instructions

1. Ensure MATLAB is installed on the machine. The version used was R2006a.

2. Ensure MATLAB is registered as a COM server. Enter the following into the DOS command window:

   cd $MATLAB\bin\win32

   matlab /regserver

   where "$MATLAB" is the MATLAB root directory. The default MATLAB root directory for R2006a is "C:\Program Files\MatLab\R2006a"

3. Ensure OPNET Modeler is installed on the machine. The version used was 12.0.

4. Since OPNET does not seem to like a path with spaces in it, you will need to copy the MATLAB library files to a path that does not have spaces. The files you will need to copy are "libmat.lib, libeng.lib, libmex.lib, and libmx.lib". These files are located at "$MATLAB\extern\lib\win32\microsoft". I copied the files to "C:\matlab_libs".

5. You will need to copy the files engine.h, tmwtypes.h, and matrix.h from "$MATLAB\extern\include" to "$OPNET\models\std\include" where "$OPNET" is the OPNET root directory. The default OPNET root directory for version 12.0 is "C:\Program Files\OPNET\12.0.A".

6. Open OPNET. Once on the main screen click on "Edit" and then "Preferences".

7. Maximize "Discrete Event Simulation", maximize "Code Generation", and then click on "Linking'.

8. Add "libmat.lib libeng.lib libmex.lib libmx.lib" to the "Common Network Repository Libraries" field.

9. In the "Common Network Repositories Flags" field, you will need to enter the path for the library files that were copied. In my case, "/LIBPATH:C:\matlab_libs" was entered into this field.

10. In your process model, you will need to place "#include "engine.h"" in the header block.

11. You will use the engine API provided by MATLAB to call MATLAB from C. First you will open a MATLAB engine through the "engOpen" command. The MX Array data structure is used to pass information between C and MATLAB. The "engPut-Variable" and "engGetVariable" commands are used either to pass an MX Array to MATLAB or to receive an MX array from MATLAB. The "engEvalString" command is used to give MATLAB commands to the MATLAB engine. Finally, the "engClose" command is used to close the MATLAB engine. A complete list of the MATLAB Engine commands and MX Array Manipulation commands are included in the MATLAB help file.

12. The following example opens a MATLAB engine, creates a MX Array, passes it to MATLAB, performs an operation on it, passes the MX Array back to C, and closes the MATLAB engine.

A.2 Example Function

```
static void
    matlab_example(void)
    {
/* declare variables*/
    Engine* ep;
    char          message[512];
    mxArray*      x;
    double            x_data = 2;
    mxArray*       y;
    double* y_ptr;
    double y_data[2*1] = {3,4};
    mxArray*      z;
    double z_data[2*1];
    double*          z_ptr;


FIN (matlab_example(void));
/* Open Matlab engine. If engine doesn't open display error message. */
if (!(ep = engOpen(NULL)))
            {
            sprintf(message, "\nCan't start MATLAB engine\n");
            op_prg_odb_print_major(message, OPC_NIL);
            }

/* Create a mx scalar array and pass it to MATLAB */
x = mxCreateDoubleScalar(x_data);
engPutVariable(ep, "x", x);
mxDestroyArray(x);
/* Create a mx matrix array and pass it to MATLAB */
y = mxCreateDoubleMatrix(2,2,mxREAL);
y_ptr = mxGetPr(y);
memcpy(y_ptr, y_data, 2*1*sizeof(double));
engPutVariable(ep, "y", y);
mxDestroyArray(y);
/* multiply two arrays in MATLAB*/
engEvalString(ep, "z = x * y");
/* Receive the result from MATLAB */
z = engGetVariable(ep, "z");
z_ptr = mxGetPr(z);
memcpy(z_data, z_ptr, 2*1*sizeof(double));
mxDestroyArray(z);
/* close the MATLAB engine */
engClose(ep);
/* Display the answer */
```

A-3

```
sprintf(message, "z1 = %f and z2 = %f", z_data[0], z_data[1]);
          op_prg_odb_print_major(message, OPC_NIL);
FOUT;
}
```

Bibliography

1. Anjali, Tricha, et al. "New MPLS Network Management Techniques Based on Adaptive Learning," *IEEE Transactions on Neural Networks*, *16*(5):1242–1255 (September 2005).

2. BAE Systems, http://www.eis.na.baesystems.com/nes/products, February 2007.

3. Bailey, N. T. J. "A Continuous Time Treatment of a Single Queue Using Generating Functions," *Journal of the Royal Statistical Society*, *16*:288–291 (1954).

4. Bletsas, Aggelos. "Evaluation of Kalman Filtering for Network Time Keeping," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, *52*(9):1452–1460 (September 2005).

5. Brakmo, Lawrence S. and Larry L. Peterson. "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, *13*(8):1465–1480 (October 1995).

6. Cantrell, P. E. "Computation of the Transient M/M/1 Queue Cdf, Pdf, and Mean with Generalized Q-Functions," *IEEE Trans. Communications*, *COM-34*:814–817 (August 1986).

7. Casetti, Claudio, et al. "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks," *Wireless Networks*, *8*(5):467–479 (2002).

8. Castro, Rui, et al. "Network Tomography: Recent Developments," *Statistical Science*, *19*(3):499–517 (2004).

9. Floyd, Sally. "TCP and Explicit Congestion Notification," *ACM SIGCOMM Computer Communication Review*, *24*(5):10–23 (October 1994).

10. Grieco, Luigi A. and Saverio Mascolo. "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," *ACM SIGCOMM Computer Communications Review*, *34*(2):25–38 (April 2004).

11. Gross, D. and C. M. Harris. *Fundamentals of Queueing Theory* (3rd Edition). New York: Wiley-Interscience, 1998.

12. Karagiannis, T., et al. "A Nonstationary Poisson View of Internet Traffic," *INFOCOM 2004. Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies*, *3*:1558–1569 (7-11 March 2004).

13. Kurose, James F. and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet* (3rd Edition). Boston: Pearson /Addison Wesley, 2005.

14. Li, Shupeng and Nirwan Ansari. "TCP-Jersey over High Speed Downlink Packet Access," *IEEE Globecom 2005*, *6*:3576–3580 (2005).

15. Mascolo, Saverio, et al. "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, 287–297 (2001).

16. Maybeck, P. S. *Stochastic Models, Estimation, and Control, I*. New York: Academic Press, Inc, 1979. Republished, Arlington, VA: Navtech, 1994.

17. Maybeck, P. S. *Stochastic Models, Estimation, and Control, II*. New York: Academic Press, Inc, 1982. Republished, Arlington, VA: Navtech, 1994.

18. Maybeck, P. S. *Stochastic Models, Estimation, and Control, III*. New York: Academic Press, Inc, 1982. Republished, Arlington, VA: Navtech, 1994.

19. OPNET website, http://www.opnet.com, Jan 2007.

20. Robertazzi, T. G. *Computer Networks and Systems* (3rd Edition). New York: Springer, 2000.

21. Song, Ci, et al. "A Link Adaptation Approach for QoS Enhancement in Wireless Networks," *IEEE Conference on Local Computer Networks (LCN)*, 373 (2001).

22. Soule, Augustin, et al. "Traffic Matrix Tracking Using Kalman Filters," *SIGMETRICS Performance Evaluation Review*, *33*(3):24–31 (2005).

23. Stallings, W. *High-Speed Networks and Internets* (2nd Edition). Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2002.

24. Stevens, W. Richard and Gary R. Wright. *TCP/IP Illustrated*, *1-3*. Reading, Mass: Addison - Wesley, 1994-1996.

25. The Mathworks - MATLAB and Simulink for Technical Computing, http://www.mathworks.com, Jan 2007.

26. Xu, Kai, et al. "TCP-Jersey for Wireless IP Communications," *IEEE Journal on Selected Areas in Communications*, *22*(4):747–756 (May 2004).

Vita

Nathan Stuckey is a student at the Air Force Institute of Technology, pursuing a Masters Degree/Electrical Engineering. His studies are in Stochastic Estimation and Control and Computer Networking. Captain Stuckey will graduate from AFIT in March 2007. Upon graduation, he will be assigned to Eglin AFB, 46th Test SQ. He completed his Bachelor's Degree in Electrical Engineering at Purdue University in 2002. He is a member of Tau Beta Pi and Eta Kappa Nu.

REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
					19b. TELEPHONE NUMBER *(Include area code)*